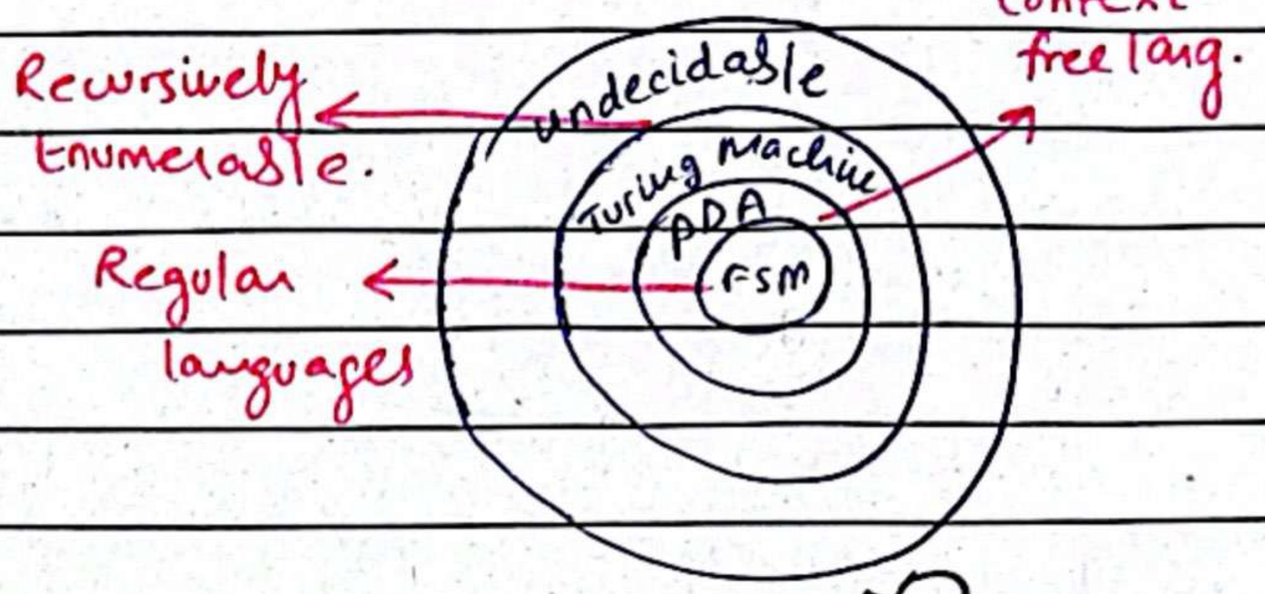
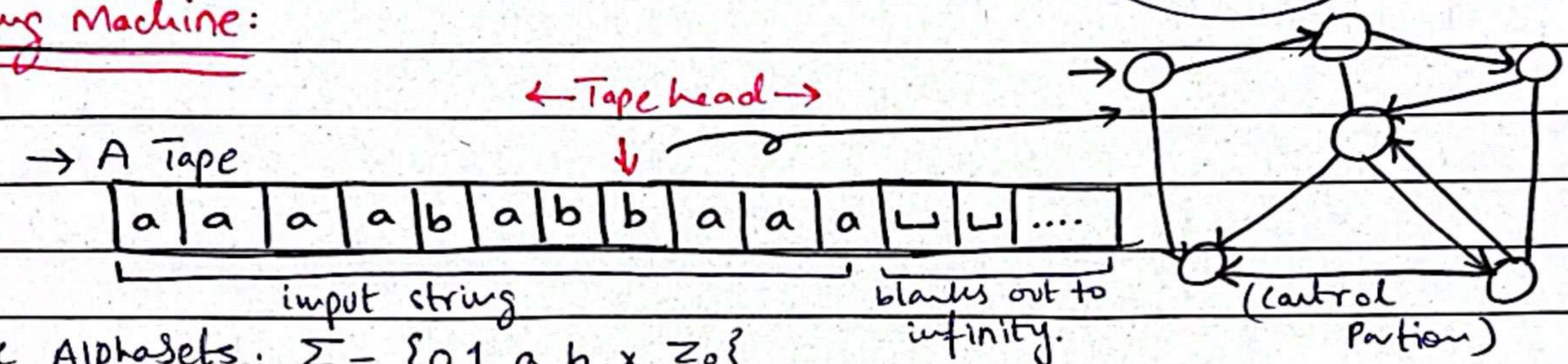
Date: _____

TURING MACHINE



FSM: The input string

Turing Machine:



Tape Alphasets: $\Sigma = \{0, 1, a, b, x, z_0\}$

The Blank ⊔ is a special symbol ⊔ ∉ Σ

The blank is a special symbol used to fill the infinite loop.

Operations on Tape:

1. Read/Scan symbol below Tape head.
2. Update/write a symbol below Tape head.
3. Move tape head one step LEFT
4. Move Tape head one step RIGHT

The control Portion similar to FSM or PDA

The control Program.

It is deterministic.

Rules of operation ①

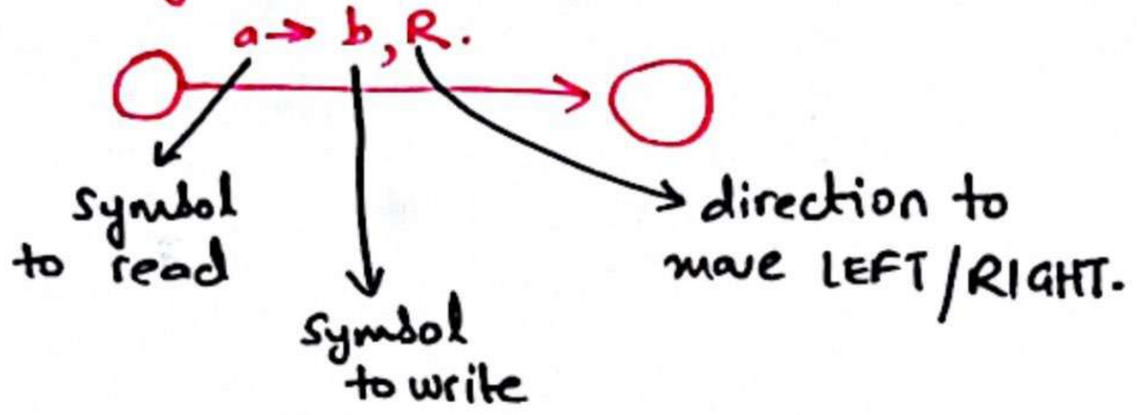
At each step of computation.

① Read the current symbol.

② Update (ie write) the same cell.

③ Move exactly one cell either LEFT/RIGHT.

If we are on left end of tape, and trying to move LEFT, then do not move. Stay at the left end.



* if you don't want to update cell, JUST WRITE SAME SYMBOL.

Rules of operation ②

→ control is with a sort of FSM.

→ initial state

→ final state

① ACCEPT ② REJECT

→ computation can either

① HALT and ACCEPT

② HALT and REJECT

③ LOOP.

*

A Turing Machine can be defined as follows.

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

$Q \rightarrow$ non empty states

$\Sigma \rightarrow$ non empty set of symbols.

$\Gamma \rightarrow$ non empty set of Tape Symbols.

$\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$

$\delta \rightarrow Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$

$q_0 \rightarrow$ initial state.

$b \rightarrow$ Blank Symbol.

$F \rightarrow$ set of final states (Accept/Reject).

Thus, the production rule of Turing Machine will be written as.

$$\delta(q_0, a) \rightarrow (q_1, Y, R)$$

Date:_____

TURING THESIS.

Turing's Thesis states that any computation that can be carried out by mechanical means can be performed by some Turing Machine.

few anguments for accepting this thesis are:

1. Anything that can be done on existing digital computer can also be done by Turing Machine.
2. No one has yet been able to suggest a problem solvable by what we ravider on algorithm, for which Turing Machine Program canot be written.

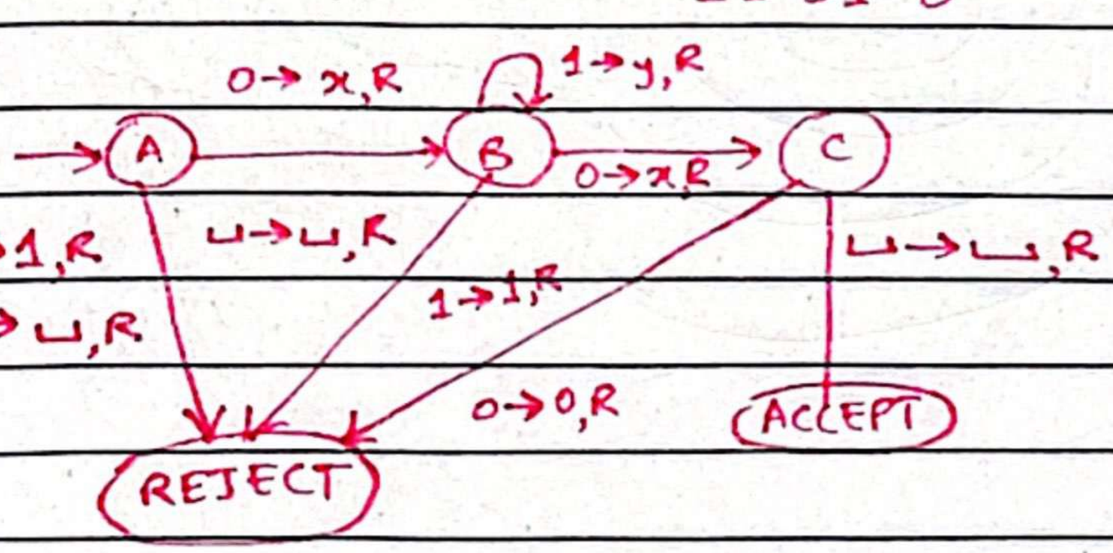
Recursively Enumerable Language.

A language L and Σ is said to be Recursiwely Enumerable if there exists a Turing Machine that accepts it.

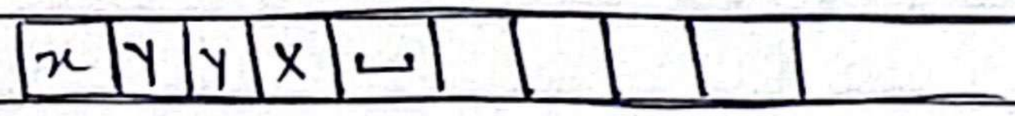
EXAMPLE (1)

a) Design a Turing Machine which recognizes the language

$L = 01^*0$



0110



Tape

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

$b = ␣$

Date:_____

EXAMPLE (2)

Q) Design Turing Machine which recognizes the language.

$L = 0^N 1^N$

0	0	0	0	1	1	1	1	⊔	⊔	...

Algorithm:

→ Change "0" to "x"

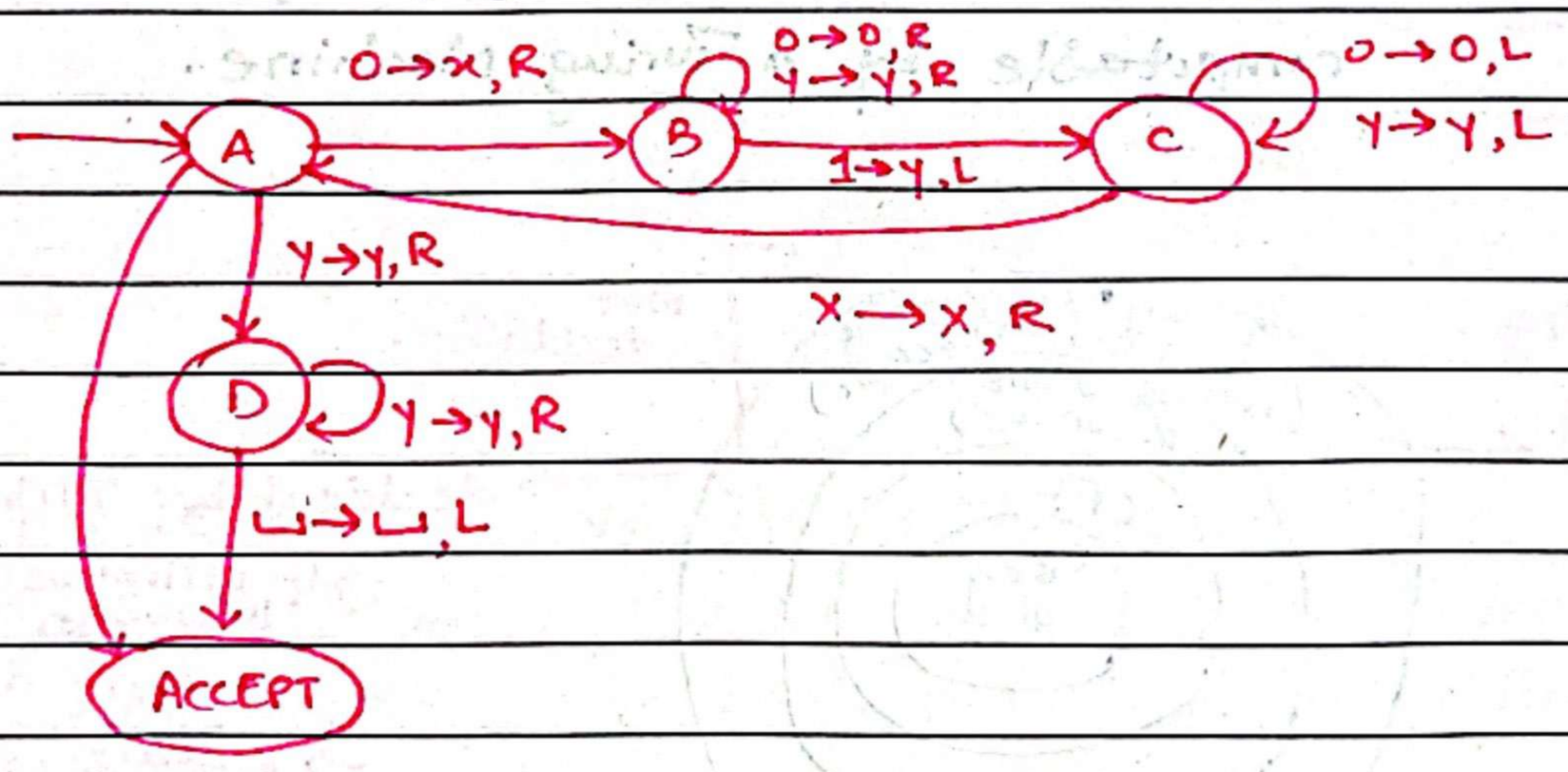
→ Move right to first "1" (if None: REJECT)

→ Change "1" to "y"

→ Move LEFT to leftmost "0"

→ Repeat above steps until no more "0s"

→ Make sure no more "1"s remain



Date:_____

The CHURCH-TURING Thesis.

Several Variations of TM:

→ One Tape or many

→ Infinite on both ends

→ Alphabets only {0,1} or more?

→ Can the Head also stay in the same place?

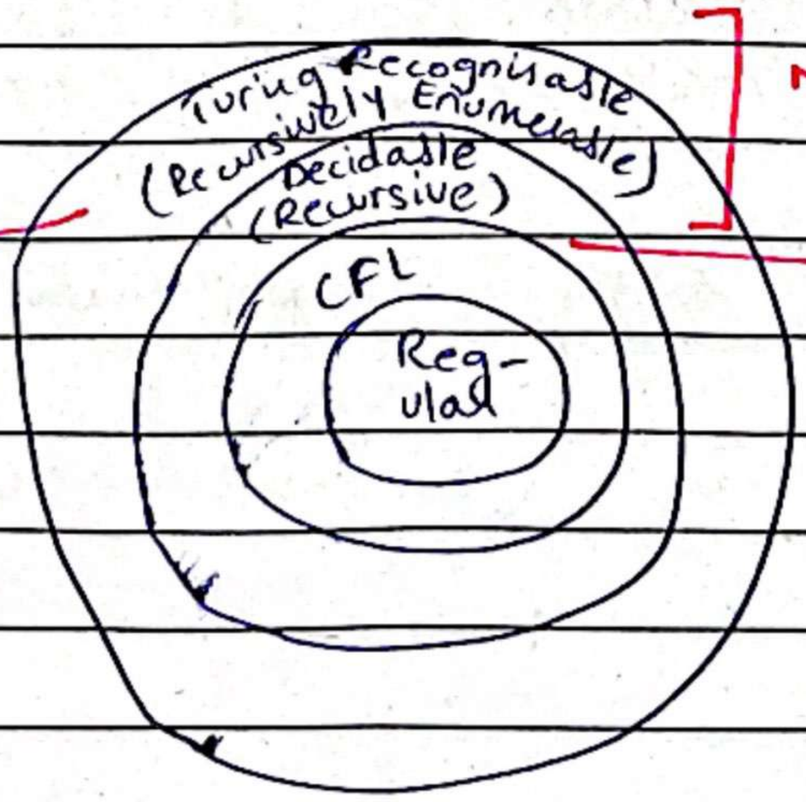
→ Allow Non-Determinism

Turing Machine & Lambda Calculus are also equivalent in power.

Algorithmically computable means.

computable by a Turing Machine.

⚠ Turing Machine ≠ Turing Test



→ if you pass string does not belong to language TM will go to LOOP

Not decidable.

→ decided by TM.

→ if string belong to language TM will accept & halt.

→ always come to HALT

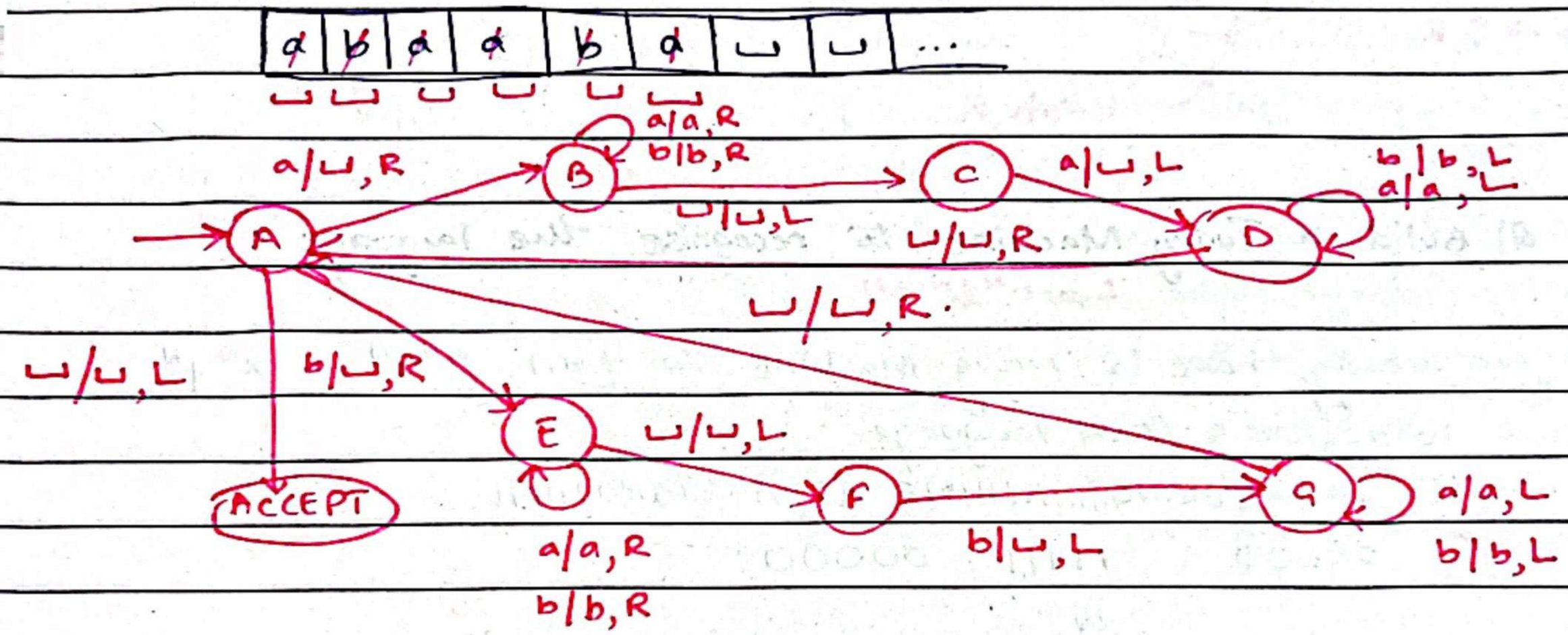
Date:_____

Turing Machine for Even Palindromes.

Q) Design a Turing Machine that accepts Even Palindromes over the alphabet.

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

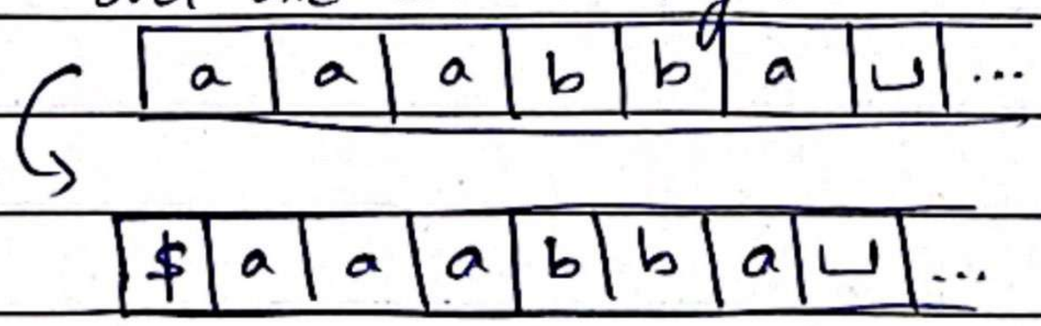
e.g. a b a a b a.



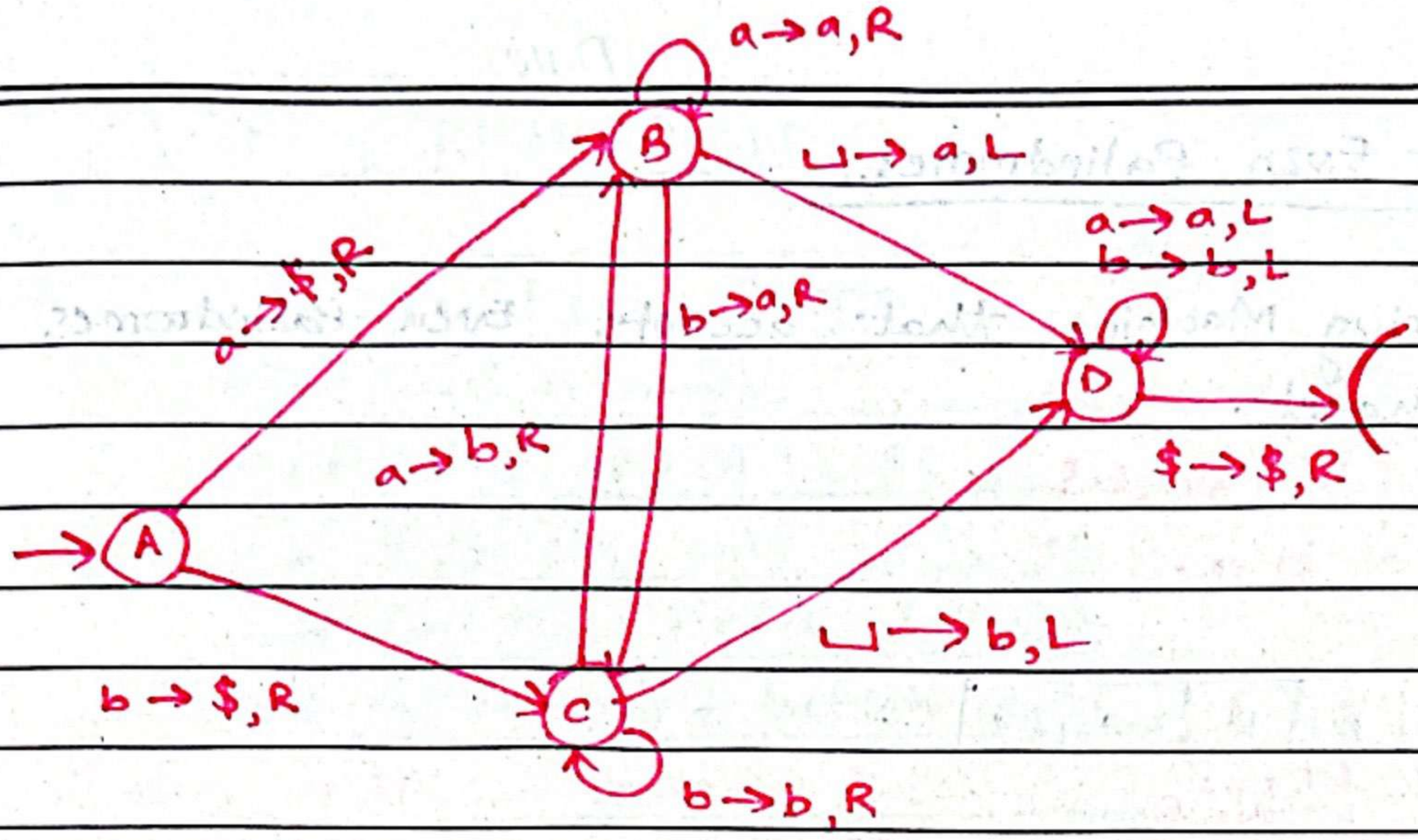
TURING MACHINE PROGRAMMING TECHNIQUES (Part 1)

Q) How can we recognise the left end of Tape of Turing Machine?

Put special symbol $ on left end of Tape and shift the input over one cell to right



Date:_____



Q) Build a Turing Machine to recognise the language

$L = 0^N 1^N 0^N$

We dready have a Turing Machine to turn $0^N 1^N$ to $x^N y^N$ and to decide that language.

USE THIS TURING MACHINE AS A SUBROUTINE

(1) 00000 11111 00000

⇓

xxxxx yyyyy 00000

(2) Build a similar Turing Machine to recognise $y^N 0^N$

(3) Build the final Turing Machine by combining these two smaller Turing Machine together into one larger Turing Machine.

Date:_____

COMPARING TWO STRINGS.

A Turing Machine to decide $\{w\#w \mid w \in \{a,b,c\}^*\}$

Solution:

→ use a new symbol such as 'x'

→ Replace each symbol into an x after it has been examied

a b b a c # a b b a c

⇓

x b b a c # x b b a c

⇓

x x b a c # x x b b a c

⇓

x x x a c # x x x a c

⇓

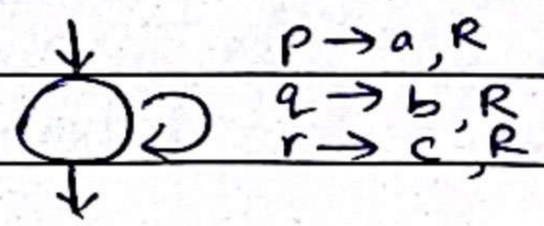
x x x x x # x x x x x

q) Can we do it non-destructively? (ie without loosing original string)

Solution

Replace each unique symbol with another unique symbol instead of replacing all with same symbol-

p q q p r # p q q p r



MULTI-TAPE TURING MACHINE

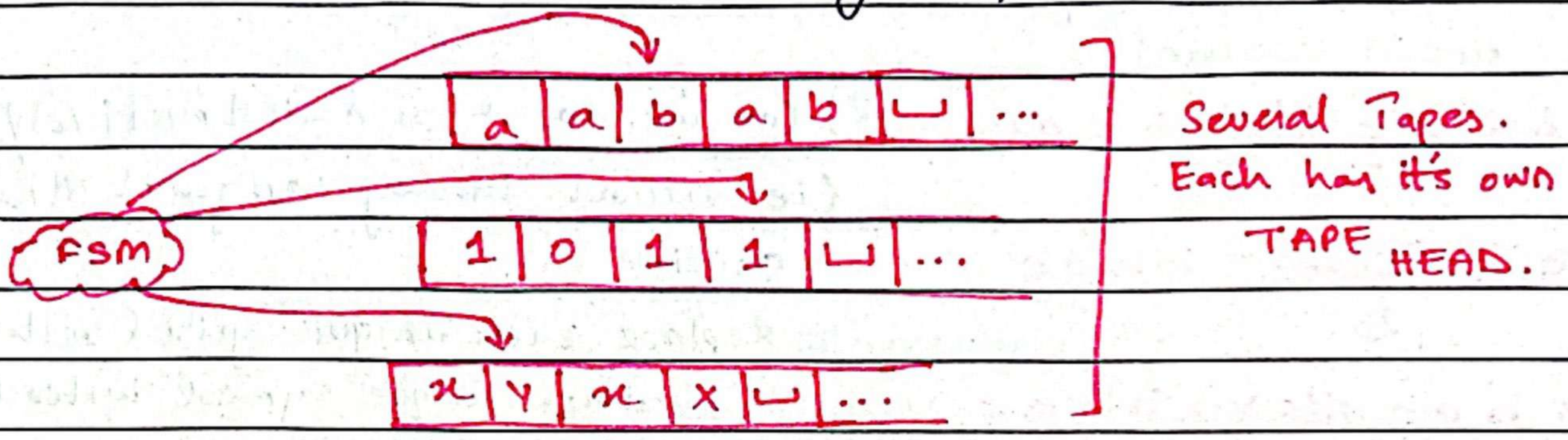
Theorem: Every Multitape Turing Machine has an equivalent Single Tape Turing Machine.

Proof:

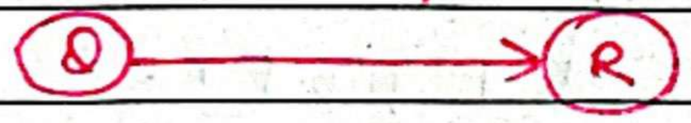
Given a Multitape Turing Machine show how to to build a single tape Turing Machine.

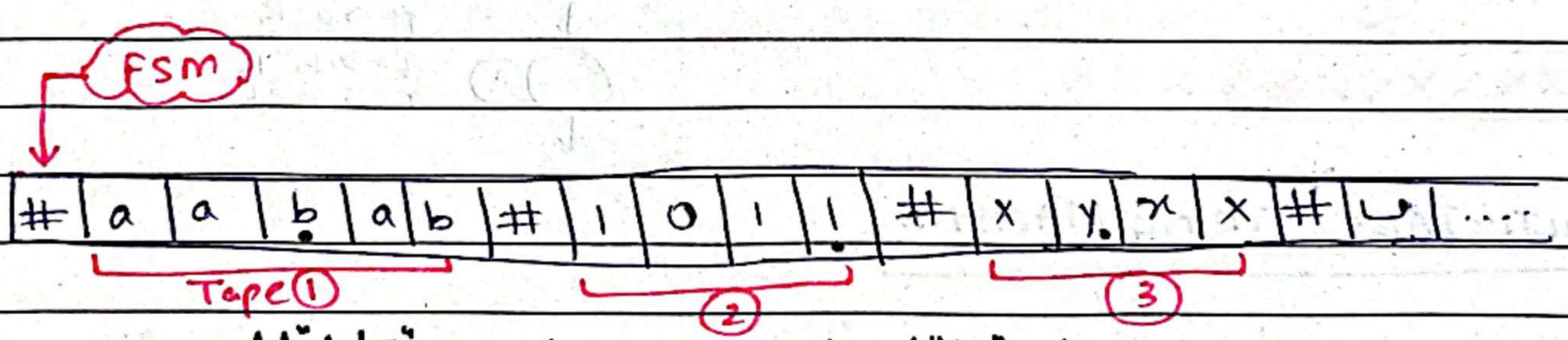
Date:_____

- Need to Store all tapes on a single tape.
  Show data representation.
- Each tape has a tape head.
  Show how to store that info.
- Need to tranform a move in Multitape TM into one or ~~more~~ moves in the Single Tape TM.



b1y → a0x, LLR.





- add "dots" to show where Head "K" is
- to simulate a transition from state Q, we must scan our Tape to see which symbols are under K Tape Heads.
- once we determine this and are ready to make the transition, we must scan across the tape again to update. cells and move dots.

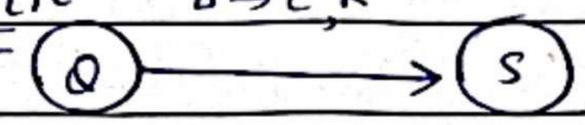
Date:_____

Nondeterminism in Turing Machine.

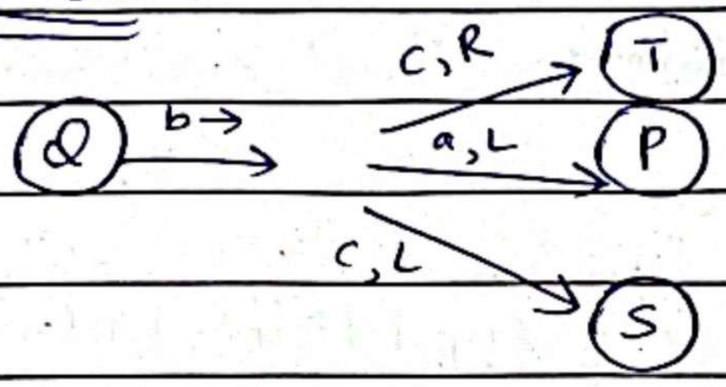
Transition function.

$$\delta : Q \times \Sigma \rightarrow P\{T \times (R/L) \times Q\}$$

Deterministic



Non-deterministic



CONFIGURATION

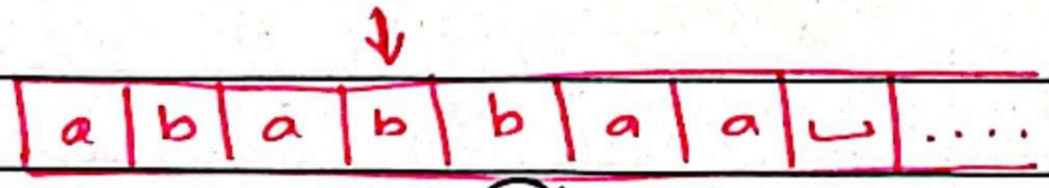
→ A way to represent the entire state of Turing Machine at a moment during computation

→ A string which captures.

→ Current state

→ Current position of Head.

→ entire Tape contents

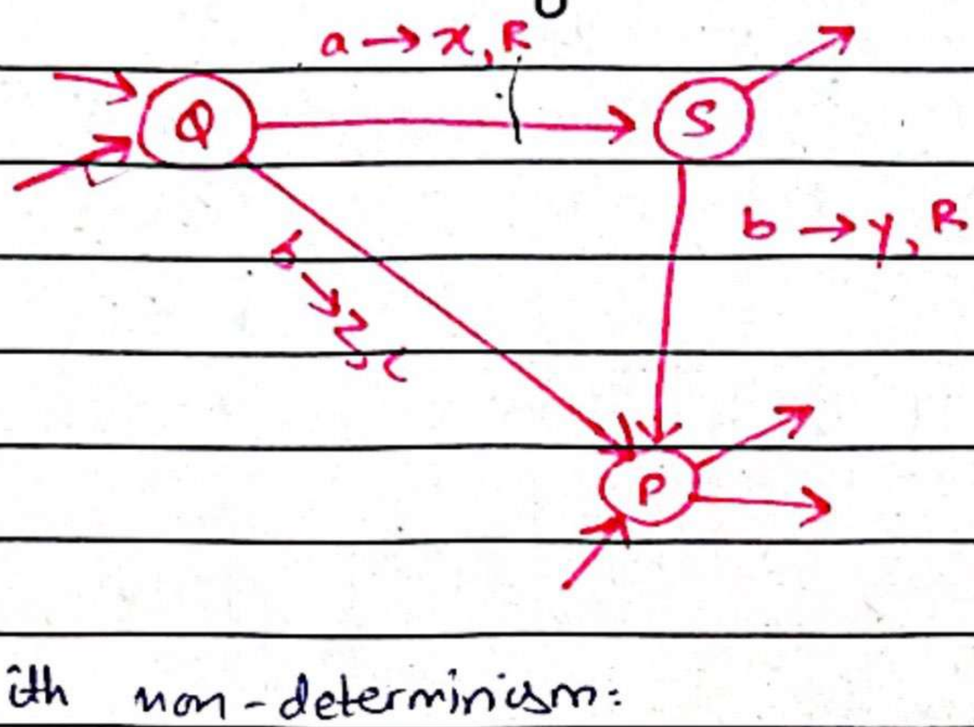


* state & looking at symbol b in tape.

* string can tell us ~~that~~ status of TM. at any particular moment.

Date:_____

Deterministic Turing Machine.



Computation History

cc Q a bb c

⇓

cc x S b b c

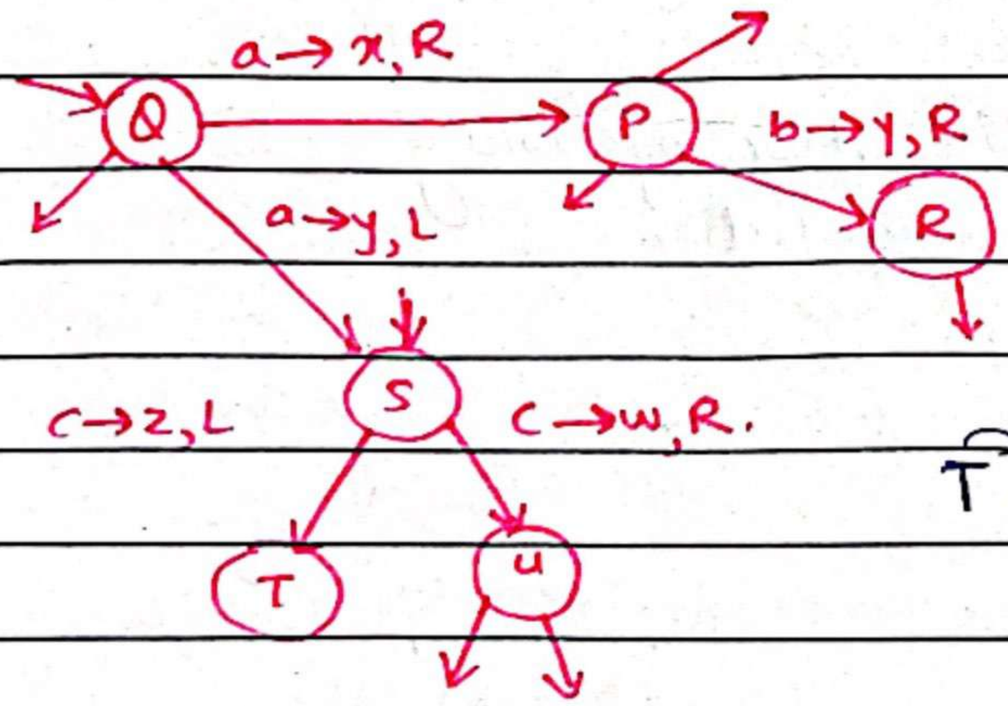
⇓

cc x yP b c

With non-determinism:

At each manet in caputation there can be mave than one successor configration.

Non-deterministic TM:



Computation History

cc Q a b b c

cc x P b bc

cS c ybbc

cc x y R b c

Tczybbc cwUybbc

Date:_____

Outcomes of a Nondeterministic Caputation.

ACCEPT: If any ~~boo~~ brauch of caputation accepts, then the nondeterministic TM will Accept.

REJECT: If all branches of the caputation HALT and REJECT (ie no branches accept, but all caputations HALT) then the Nondeterntic TM Rejects.

LOOP: Caputation cantinues but ACCEPT is neves uncountced Sane branches in caputation history are infinite.

THEOREM: (PROOF).

Every nondeterministic TM has an equivalent Deterministic TM.

→ The pawer of NTM and DTM is sare.

Proof:

→ Given a Non-deterministic TM (N) shaw haw to cantruct an equivalent Deterministic TM (D).

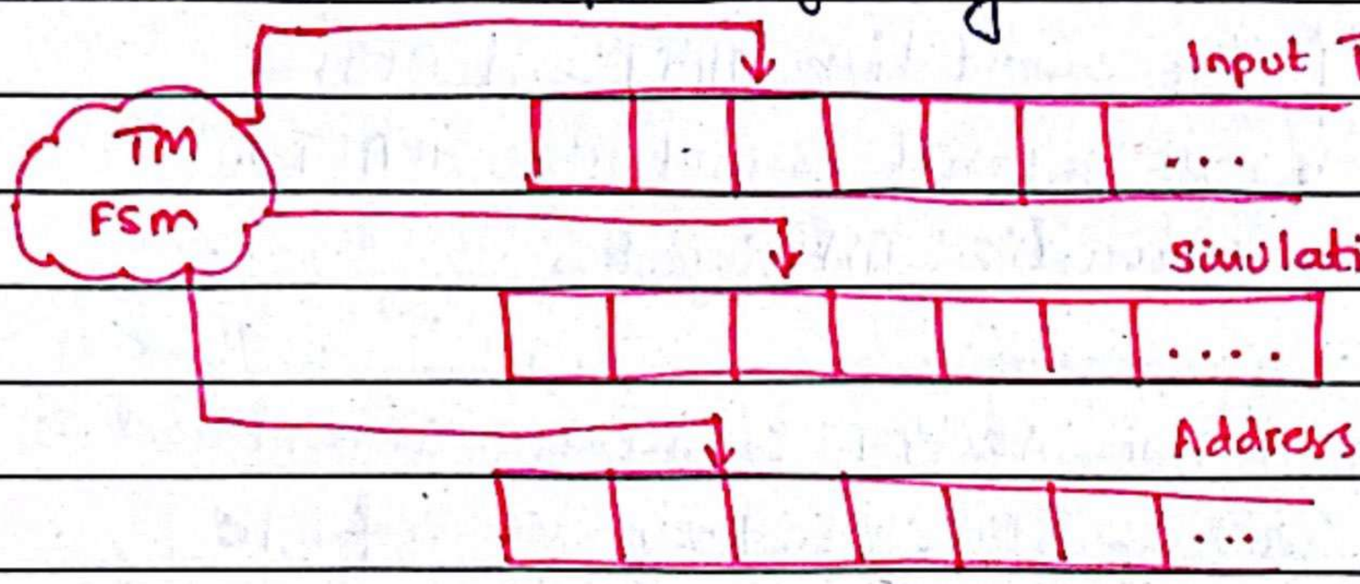
→ if N accepts on any branch, the D will Accept.

→ if N halts an every branch without any ACCEPT, then D will Halt & Reject.

Date: _____

Approach:

1. Simulate N
2. Simulate all branches of computation.
3. Search for any way N can accept.



Algorithm

Initially: Tape 1 contains the input
Tape 2 and Tape 3 are empty

→ Copy Tape 1 to Tape 2.
→ Run the simulation
→ Use Tape 2 as "the Tape"
→ When choices occur consult Tape 3.
→ Tape 3 contains a Path. Each number tells which choice to make
→ Run the simulation all the way down the branch as far as the address/path goes (or computation dies)
→ Try the next branch.
→ Increment the addresses on Tape 3.
→ REPEAT.

* if ACCEPT is ever encountered, Halt and Accept
* If all branches Reject or die out, the Halt & Reject.

CHAPTER 4: DECIDABILITY.

Date:_____

Recursive language:

→ A language 'L' is said to be recursive if there exists a Turing Machine which will accept all strings in 'L' and reject all strings not in 'L'.

→ The turing machine will halt every time and give an aswer (accepted or rejected) for each and every string input. will always HALT (Accept & halt, Reject & Halt)

Recursively Enumerable Language:

→ A language 'L' is said to be recursively enumerable language if there exists a TM which will accept (and therfore halt) for all the input strings which are not in L.

→ But may or may not halt for all inputs strings which are not in L.
(Accept & halt).

Decidable Languages:

A language 'L' is said to be decidable if it is a recursive language. All decidable languages are recursive languages and vice vessa.

Date:_____

Partially Decidable:

A language 'L' is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language:

→ An undecidable language may sometimes be partially decidable but not decidable.

→ If a language is not even partially decidable, then there exists no TM for that language.

Recursive language → TM will always HALT

Recursively Enumerable " " → TM will halt sometimes & may not halt sometimes.

Decidable language → Recursive language

Partially Decidable language → Recursively Enumerable

Undecidable → No TM for that language.

The Universal Turing Machine.

The language.

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a Turing Machine and M accepts $w \}$

is Turing Recognisable.

Date:_____

Given a the description of a TM and sane input
can we determine whether the machine accepts it?
→ Just Simlate/Run the TM on the input.

M accepts w: Our Algorithm will Halt & Accept
M rejects w: " " Halt & Reject
M loops w: " " will not Halt.

Input M = description of TM, w = an input string for M.
Action ① Simulate M
② Behave just like M would (may accept/reject/loop)

★ Computer is like universal Turing Machine
→ difference
→ TM has infinite tape
→ But computer can never have tape/memory infinite.

The UTM is a recognizer (but not decider).

HALTING PROBLEMS
Given a program: WILL IT HALT?

Date:_____

Given a Turing Machine, will it halt when run on some particular given input?

Undecidability of Halting Problem.

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

We will prove by Contradiction:

Let us anume that we can:

H (P, I)

- Halt
- Not halt.

This allows us to write another Program:

```
C(X).
  if {H(X,X) == Halt}
        loop forever;
  else
        return;
```

Date: _____

DECIDABILITY & UNDECIDABILITY PROBLEMS.

→ The "Halting Problem" is not decidable.

The "HALTING PROBLEM"

Given a program... WILL IT HALT?

Given a Turing Machine, will it halt when run on some particular given input string?

Given some program written in [Java | C | any other language], will it ever get into an infinite loop? or will it always terminate?

ANSWER

In general, we can't always know!

The best we can do is run the program and see whether it halts ... BUT...

For many programs, we can prove "It will always halt". [OR "It may sometimes LOOP"]

But for programs in general the problem is UNDECIDABLE

PROBLEM (1)

Theorem.

The language ...

$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \}$

... is decidable.

Provide a TM D_{A-DFA} that decides A_{DFA}. The TM is given as input $\langle B, w \rangle$, a DFA and a string w.

Date:_____

① Checks that it has the form <B, w> where B is DFA and w is a string.
② Simlate the computation of B on w.
③ If B ends in an accept state then accept. If not then reject.
This TM will always HALT.

PROBLEM ②

Theorem.

The language...

$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is a NFA and } B \text{ accepts } w \}$.

... is decidable.

Construct a TM D_{A-NFA} that takes as input representation of an NFA ^{B} and a string w.

① Convert NFA B to equivalut DFA B'
② Run TM D_{A-NFA} on input $\langle B', w \rangle$
③ Accept if simulation accepts otherwise reject.

PROBLEM ③

Theorem.

The language...

$A_{REX} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates string } w \}$

... is decidable.

We can build a TM / We can write a program that given a reg expression R and string w as input will determine $\langle R, x \rangle$ is an A_{REX}.

$R = 0^*1^*$

$S = 0^*1^*$

Date:

AND

that algorithm / TM will always HALT?

Step (1) Convert R into a NFA B'

(2) construct the string $\langle B', w \rangle$

PROBLEM (4)

Theorem.

The language ...

$E_{DFA} = \{ \langle B \rangle \mid B \text{ is a DFA and } L(B) = \phi \}$

... is decidable.

Give TM $D_{E\text{-}DFA}$ that decides E_{DFA}.

(1) Mark start/initial state.

(2) Repeat until no new state is marked:

Mark every state that has incoming arrow from previously marked state.

(3) Accept if no accept state is marked

Reject if some accept state is marked.

PROBLEM (5)

Theorem

The language ...

$EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ \& } B \text{ are DFA and } L(A) = L(B) \}$.

... is decidable.

R

Date:_____

Let C be the "SYMMETRIC DIFFERENCE" b/w A and B.

Given A = DFA to Accept L(A)

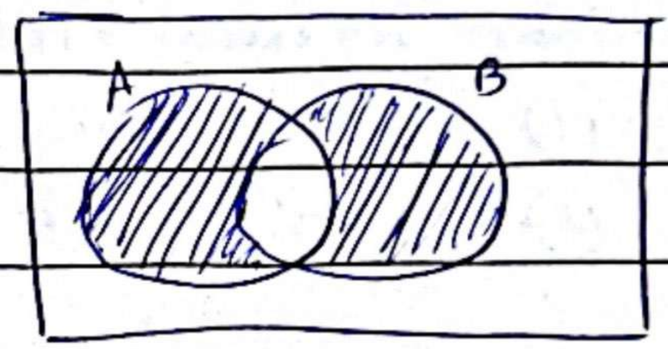
B = DFA to Accept L(B)

.... we know how to combine DFAs.

$\overline{L(A)}$

$L(A) \cup L(B)$

$L(A) \cap L(B)$



$C = (A \cap \bar{B}) \cup (\bar{A} \cap B)$

Build DFA C to accept the symmetric difference.

Use the TM from previous theorem [E_{DFA} = empty language] to test

——— X ——— X ——— X ——— X PROOF:

Construct a TM

Input: <A,B> (2 DFA's)

Construct a DFA C to accept

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

Use The previous TM to test whether the language that C accepts is empty.

Accept if so, otherwise reject.

PROBLEM (6)

Theorem.

The language...

$A_{CFG} = \{ <G, w> \mid G \text{ is a CFG and } w \in L(G) \}$.

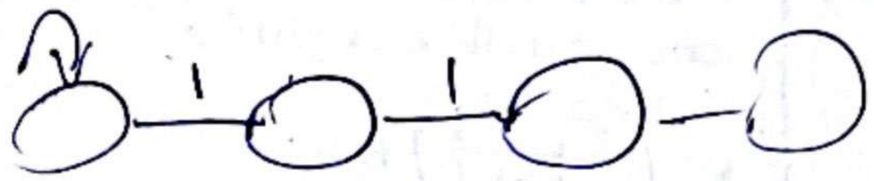
... is decidable

(1) Convert G into CNF.

(2) Try all derivations of length $2|w|-1$

(3) Accept: if any generate w.

Reject: if not.



Date:_____

PROBLEM (7)

Theorem.

The language ...

$E_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \phi \}$.

... is decidable.

(1) Mark all occurences of terminals in G.

(2) Repeat until no new variables are marked.

Mark all occurences of variable A if

$A \to B_1 B_2 \dots B_k$ is a rule and all B_i were already marked.

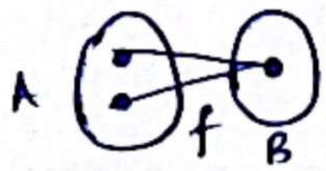
(3) Reject: if the start variable is marked.

Accept: if not.

DEFINITIONS Assume $f: A \to B$

"ONE TO ONE"

If $a \neq b$ then $f(a) \neq f(b)$



ONTO

every element in B is "Hit"

"CORRESPONDENCE"

ONE TO ONE and ONTO

Date: _____

THE UNIVERSAL TURING MACHINE.

THEOREM

The language...

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a Turing Machine and M accepts $w \}$.

Is turing recognizable.

The following TM U recognizes A_{TM}

1. ~~Simulation~~ Simulate M on input w.
2. Accept if M halts and accepts.
3. Reject if M halts and rejects.

INFINITY: COUNTABLE & UNCOUNTABLE.

George Cantor: "Two sets have the same size iff there exists a correspondence b|w them"

A set is "countable" iff it has finite size or there is correspondence with $\mathbb{N} = \{1, 2, 3 \ldots\}$

ODD Numbers $= \{1, 3, 5, \ldots\}$

countably infinite also a subset of $\mathbb{N}$

ODD		$\mathbb{N}$
1	———	1
3	———	2
5	———	3
7	———	4
9	———	5
⋮		⋮

Date: _____

Rational Numbers = $\{ M/N \mid M \text{ and } N \in \mathbb{N} \}$

Countably infinite.

Irrational Numbers / Real Numbers.

Uncountably Infinite.

THE SET OF RATIONAL NUMBERS IS COUNTABLY INFINITE

PROOF: Find a way to list them must include all of them.

RATIONALS = $\{ M/N \mid M \text{ and } N \in \mathbb{N}$

CORRESPONDENCE

1/7	4/3	22/29	1/2	17/3	4/2	...
↓	↓	↓	↓	↓	↓	
1	2	3	4	5	6	...

	1	2	3	4	5	6	7	...
1	1/1	1/2	1/3	1/4	1/5	1/6	1/7	
2	2/1	2/2	2/3	2/4	2/5	2/6	2/7	
3	3/1	3/2	3/3	3/4	3/5	3/6	3/7	
4	4/1	4/2	4/3	4/4	4/5	4/6		
5	5/1	5/2	5/3	5/4	°			
6	6/1	6/2	6/3	°				
7	7/1	7/2	°					
°								
°								
°								

Date:_____

THE SET OF IRRATIONAL NUMBERS
IS UNCOUNTABLY INFINITE!

$\pi = 3.14159265...$

$\sqrt{2} = 1.4142135...$

$e = 2.7182818 28...$

$= 5.67932043...$

$1/3 = 0.333,333,33\overline{3}$

$.333,334,0\overline{0}$ < $.333,333,571,29...$

?? lecture 34/45

PROOF BY CONTRADICTION

PROOF: Assume it is countably infinite

1	.3 1 4 1 5 9 ...	PROOF BY DIAGONALIZATION
2	.1 4 1 4 2 1 ...	
3	.2 7 1 8 2 8 ...	
4	.5 6 7 9 3 2 ...	
5	.7 4 2 5 3 1 ...	
6	.3 9 2 4 5 0 ...	
	↓ ↓ ↓ ↓ ↓ ↓	

↑ .4 .5 2 8 4 1 ... * to make it differ from diagnol values by adding 1 to it.

this number is NOT in table

hence it's uncountably infinite.

← Check yellow page.

no of TM is countably infinite.

Date: _____

LANGUAGES THAT ARE NOT RECOGNIZABLE.

WE CAN ENUMERATE THE SET OF ALL TURING MACHINE

APPROACH #1

* Every Turing Machine can be encoded into a string (of finite length)
* every string of 1's and 0's is either a valid TM representation or not
* generate all strings, one after the other.
* check to see if its a valid TM.

APPROACH #2.

The alphabets are finite

There are a finite # of kinds of transitions.

$a \rightarrow b, R$ (→)

FOR $i = 1, 2, 3, \cdots$

There is a finite number of directed graph with i nodes

There is a finite number of ways to label the edges generate them all.

END.

Date:_____

THEOREM

The set of all infinite length strings over {0,1} is uncountably infinite.

PROOF (By diagonalization).

Assume the set of infinite binary strings can be enumerated (ie correspondence with $\mathbb{N}$)

$.011010111100\cdots$

Assume countably infinite.

1—	(0)	0	0	0	0	0	1	0	0	...
2—	1	(1)	1	0	0	0	1	1	0	...
3—	1	0	(1)	0	1	0	1	0	1	
4—	1	1	1	(1)	1	1	1	1	1	
5—	0	0	1	1	(0)	1	0	0	1	
6—	1	0	0	0	0	(1)	0	1	0	
7—	0	0	1	1	1	0	(0)	0	1	
8—	1	1	0	0	1	1	0	(0)	1	
	1	0	0	0	1	0	1	1	...	

is an infinite length binary string which is not equal to any string on this list!

Hence this contradicts our assumption that all infinite binary strings can be listed. Hence the set of all infinite binary strings is uncountable.

∴ Size is larger than size of set of $\mathbb{N}$

list of length 0: {ε}
length 1: {a, b}
length 2: {aa, ab, ba, bb}
length 3: {aaa, aab, aba, ... }

→ for strings of length n, there are 2^n possible strings.
→ construct one-to-one correspondence
ε → 1 aa → 4
a → 2 ab → 5
b → 3 ⋮
∴ shows one to one correspondence so it is countable.

Date:

(1)

The set of all finite length strings over $\Sigma = \{a, b\}$ is countable.

(ε) a (b) (ab) ba aa (bb) (aaa) & aab ... countably infinite.

A language that contains some of these strings & not others

A language can be fully specified by giving an infinite length binary string.

1 0 1 1 0 0 1 1 0 ...

There are uncountably many infinite length binary strings.
Therefore, the number of languages is UNCOUNTABLY INFINITE!

THEOREM
The set of all Turing Machines is countably infinite.
COROLLARY: The set of all Turing Recognizable languages is countably infinite.

THEOREM
The set of all languages is uncountably infinite.
COROLLARY: Some languages are not Turing Recognizable.

Date

The Undecidability of the Halting Problem

THE HALTING PROBLEM.

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \}$.

↑ This set is undecidable.

PROOF (PROOF BY CONTRADICTION)

Assume A_{TM} is decidable.

Let H be the algorithm/TM that decides A_{TM}.

$$H(\langle M, w \rangle) = \begin{cases} \text{ACCEPT if } M \text{ accepts } w \\ \text{REJECT if } M \text{ does not accept } w \text{ or } M \text{ will loop} \end{cases}$$

Using H, construct a new machine D ⟶ Devil Machine

Try to run D

Find a contradiction.

∴ H = A decider for Halting Problem can not exist.

PROOF (in detail)

Input To D: $\langle M \rangle$ = The description of a TM, M.

Action:

→ H(M, <M>)

Run H as a subroutines ask whether machine M would accept if given a description of itself as an input.

output Do the opposite of what H does.

H accepts ⟹ REJECT

H reject ⟹ ACCEPT

$$D(\langle M \rangle) = \begin{cases} \text{ACCEPTS, if } M \text{ does not accept } \langle M \rangle \\ \text{REJECTS, if } M \text{ accepts } \langle M \rangle. \end{cases}$$

D accepts <M> iff M doesn't accept <M>
D accepts <D> iff D deesn't accept <D>

Date:_____

NOW RUN D ON ITSELF

$$D(\langle D \rangle) = \begin{cases} \text{ACCEPT, if D does not accept } \langle D \rangle \\ \text{REJECT, if D accepts } \langle D \rangle \end{cases}$$

H accepts <M,w> exactly when M accepts w.
D rejects <M> exactly when M accepts <M>.
D accepts <D> exactly when D accepts <D>.

$\overline{A_{TM}}$ is T-unrecognizable.

"If a language L is Turing Recognizable and it's complement $\bar{L}$ is also Turing Recognizable, Then L is decidable"

Construct TM T to decide L.

want to decide L? → T= "on input x

Let M_1 be recognizer for L

Let M_2 be the recognizer for $\bar{L}$

Run M_1 and M_2 in parallel.

x is in either L or $\bar{L}$

Either M_1 and M_2 (or both)

will eventually halt and are of them will accept

If M_1 accepts then accept

If M_2 accepts, then reject"

either way | you'll eventually halt and get decision.

Date:_____

THEOREM

A language L is decidable iff L and $\bar{L}$ are Turing Recognizable.

A_{TM} is Turing Recognizable. E_{TM} is Not Recogn

A_{TM} is not decidable/undecidable E_{TM} no

Therefore: $\bar{A}_{TM}$ is NOT TURING RECOGNIZABLE.

CHAPTER 5: REDUCIBILITY.

(A technique to prove undecidability)

How can we prove that some problems are undecidable?

New Technique

"REDUCE" one problem to another problem.

Results

→ The equivalence of two Turing Machine? ... UNDECIDABLE!

→ Is a given program/algorithm guaranteed to HALT? ... UNDECIDAB

→ Will a given TM accept any string? ... UNDECIDABLE!

$\mathbb{R}$ is uncountable — Diagonalization.

Let $\mathbb{R}$ = all real numbers

Theorem: $\mathbb{R}$ is uncountable.

Proof by contradiction via diagnolization

Assume $\mathbb{R}$ countable.

So there is 1 – 1 correspondence

$f: \mathbb{N} \longrightarrow \mathbb{R}$

n	$f(n)$
1	0.123456
2	0.987654
3	0.111111
4	0.500000
5	0.314159
6	
7	
⋮	

now we construct a new number

$x = 0.x_1x_2x_3x_4x_5$ such that

- the first digit of $f(1)$ is 1 so $x_1 = 2$
- the second digit of $f(2)$ is 8 so $x_2 = 9$
- the third digit of $f(3)$ is 1 so $x_3 = 2$
- the fourth digit of $f(4)$ is 0 so $x_4 = 1$
- the fifth digit of $f(5)$ is 4 so $x_5 = 5$

$x = 0.29215$

since x differs from every $f(n)$ it cannot be in list. This contradicts our assumption that all real numbers can be listed

Hence set of real numbers is UNCOUNTABLE.

Date:

LOGIC

Known Fact A_{TM} is Undecidable.

What about some other problem P...
Is P undecidable?

THEOREM: P is undecidable.

PROOF APPROACH

→ Assume P is decidable
→ Reduce A_{TM} (a "hard" problem) into P (the "Easier" Problem)
→ Use solution of P to solve A_{TM}
Use decidability of P to find an algorithm to decide A_{TM}.
Build a TM to decide A_{TM} using the TM to decide P as a subroutine
→ But we know that a decidel for A_{TM} cannot exist.
→ Contradiction: P is not decidable!

Date:_____

HALTING PROBLEM : A proof By REDUCTION.

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ halts on } w\}$.

Recall Theorem : $HALT_{TM}$ is undecidable.

Proof by contradiction, showing that A_{TM} is reducible to $HALT_{TM}$.

PROOF

→ Assume it is decidable

There is a Turing Machine R that decides $HALT_{TM}$.

→ Use R to build another Turing Machine S that decides A_{TM}

Reduce A_{TM} to $HALT_{TM}$

→ But A_{TM} is undecidable.

S = "On input $\langle M, w\rangle$

① use R to test if M on w halts. If not, reject.

② simulate M on w until it halts

③ — if M has accepted then accept
— if M has rejected then reject

TM S decides A_{TM}, a contradiction. Therefore $HALT_{TM}$ is undecidable.

M_w works like M except
-that it always reject strings
x where $x \neq w$

Date:_____

E_{TM} is Undecidable.

The language

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \phi\}$

is undecidable reducible

Show that A_{TM} is ~~decidable~~ to E_{TM}.

PROOF:

Assume that E_{TM} is decidable and show that

A_{TM} is decidable.

Let TM R decide E_{TM}

Construct TM S deciding A_{TM}

S = " On input $\langle M, w \rangle$

1. Transform M to new TM M_w = " On input x

1. if $x \neq w$, reject

2. else run M_w on w

3. Accept if M accepts

2. Use R to test whether $L(M_w) = \phi$

3. if YES [so M rejects w] then reject

if NO [so M accepts w] then accept

Date:_____

MAPPING REDUCIBILITY.

A "computable" function:

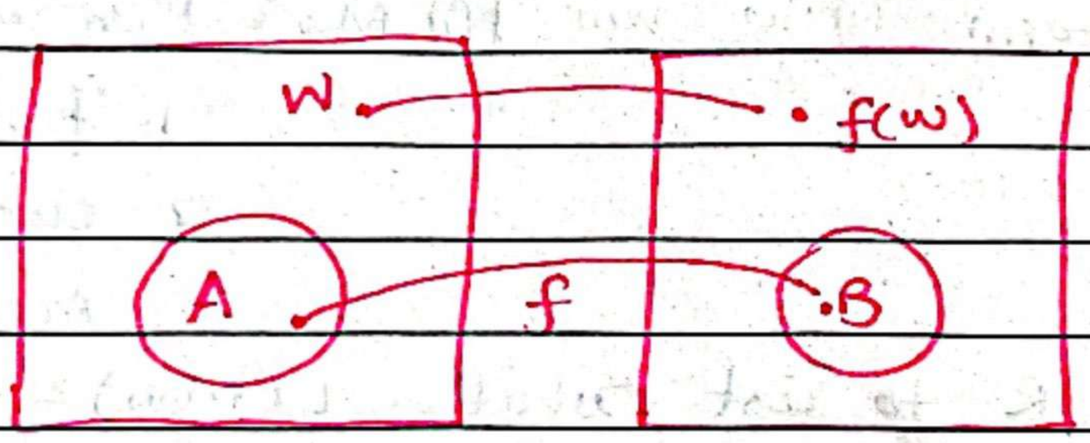
A function

$$f: \Sigma^* \rightarrow \Sigma^*$$

that can be computed by a Turing Machine.

It is computable if there is TM F where F on input w halts with f(w) on it's ~~tape~~ tape, for all strings w.

A is mapping-reducible to B ($A \leq_m B$) if there is a computable function f where $w \in A$ iff $f(w) \in B$



Theorem: If $A \leq_m B$ and B is decidable then so is A.

Proof: say TM R decides B

Construct TM S deciding A

S = " On input w

1) Compute f(w)
2) Run R on f(w) to test if $f(w) \in B$
3) If R halts then output same result

Date:_____

Corollary: If $A \leq_m B$ and A is undecidable then so is B

Theorem: If $A \leq_m B$ and B is T-recognisable then so is A

Same Proof as above.

Corollary: If $A \leq_m B$ and A is T-unrecognisable then so is B.

Mapping Reducibility useful to prove T-unrecognizability.
Reducibility useful to prove undecidability.

E_{TM} is T-unrecognizability.

$E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \phi\}$

Theorem: E_{TM} is T-unrecognizable

Show $\overline{A_{TM}} \leq_m E_{TM}$

CHAPTER 7: TIME COMPLEXITY

Date:

→ Consider only computable functions
→ decidable (always halts)
→ Consider only deterministic machines
→ Consider same input w
for TM: just count the transitions.
→ Consider all inputs of size N

Our goal: find a function of N to describe running time.

$f(N) = \ldots$

Let M be deterministic Turing Machine that always halt.
Let n be the size of an input.

Definition:
The "time complexity" of M is a function f
$f(n)$ = the maximum no of steps that M takes on any input of size n.

Note: "size of input" may mean length of string.

Algorithm to decide $\{0^k 1^k \mid k \geq 0\}$

→ Scan input to make sure it is in the form $0^* 1^*$
n steps to scan
n steps to reposition to left end.
2n steps $O(n)$.

Date: _____

→ Repeat while tape contains atleast one 0 and atleast one 1.

→ Scan across tape and change at 0 To X and 1 to X. — 2n steps O(n). } O(n)

→ end loop

n/2 repetitions.

whole loop take n/2 · O(n) → O(n²)

→ if tape contains all Xs, then accept else reject.

n steps to O(n).

O(n) + O(n²) + O(n) ⇒ O(n²). ✓

so {a^k 1^k | k ≥ 0} ∈ Time(n²)

steps to decide A = { a^k b^k | k ≥ 0 }

Theorem: A 1-tape TM M can decide A where on inputs of length n M uses at most cn² steps.

M = "On input w

O(n) 1. Scan input to check if w ∈ a*b*, reject if not

O(n) 2. Repeat until all crossed off.

O(n) { Scan tape, crossing off one a and one b
Reject if any a's or only b's remain.

O(n)+O(n²) 3. Accept if all crossed off.

⇒ O(n²).

Date:_____

Deciding $A = \{a^k b^k \mid k \geq 0\}$ faster.

Theorem. A 1-tape TM M can decide A ~~where~~ on inp by using $O(n \log n)$ steps.

M = "On input w

1. Scan tape to check if $w \in a^*b^*$
2. Repeat until all crossed off
   Scan tape, crossing off every other a and b
   Reject if even/odd parities disagree.

More precisely

$f(n) = O(g(n))$

if $\exists c$ and $\exists n_0$ such that

$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

———x———x———x———

But there is better algorithm!

→ Scan the input to make sure it is in form 0^*1^* $O(n)$

→ Repeat while the tape contains at least one 0 and atleast one 1 ...

$O(n)$ → Scan tape to see if number of 0's plus number of 1's is ODD or EVEN. $O(n)$

→ if odd then reject $O(1)$

$O(n)$ → Scan across the entire tape cross off every other 0 starting with first 0

Cross off every other 1 starting with first 1

→ End → $(1 + \log_2 n) \cdot O(n) = O(n \log n)$

Date:_____

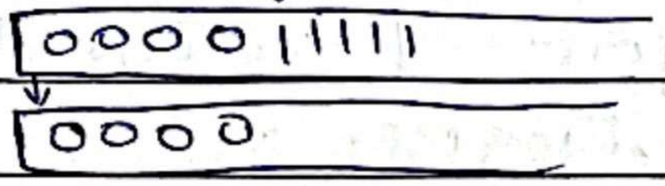
→ If no 0s and no 1s remain then ACCEPT, else REJECT $O(n)$.

So $\{0^k1^k \mid k \geq 0\} \in$ TIME $(n \log n)$

DIFFERENT MODELS OF COMPUTATION.

Algorithm using 2 tapes. $\{0^k1^k \mid k \geq 0\}$

→ copy all 0's to tape 2 → $O(n)$.



→ Reposition tape 2 to beginig → $O(n)$

→ Scan both tapes siwltaneously

→ Make sure both tape heads hit the ⌴ at same time → $O(n)$.

Theorem. For every multitape Turing Machine algorithm that takes time $t(n)$, There is an equivalent single tape Turing Machine that takes time $O(t^2(n))$

PROOF:

In time $t(n)$, the largest the tapes can be is $t(n)$. You can siwlate the multitape algorithm on a machine with one tape. Each step of siwlation. can be done in $O(t(n))$ time. To siwlate the entire algorithm

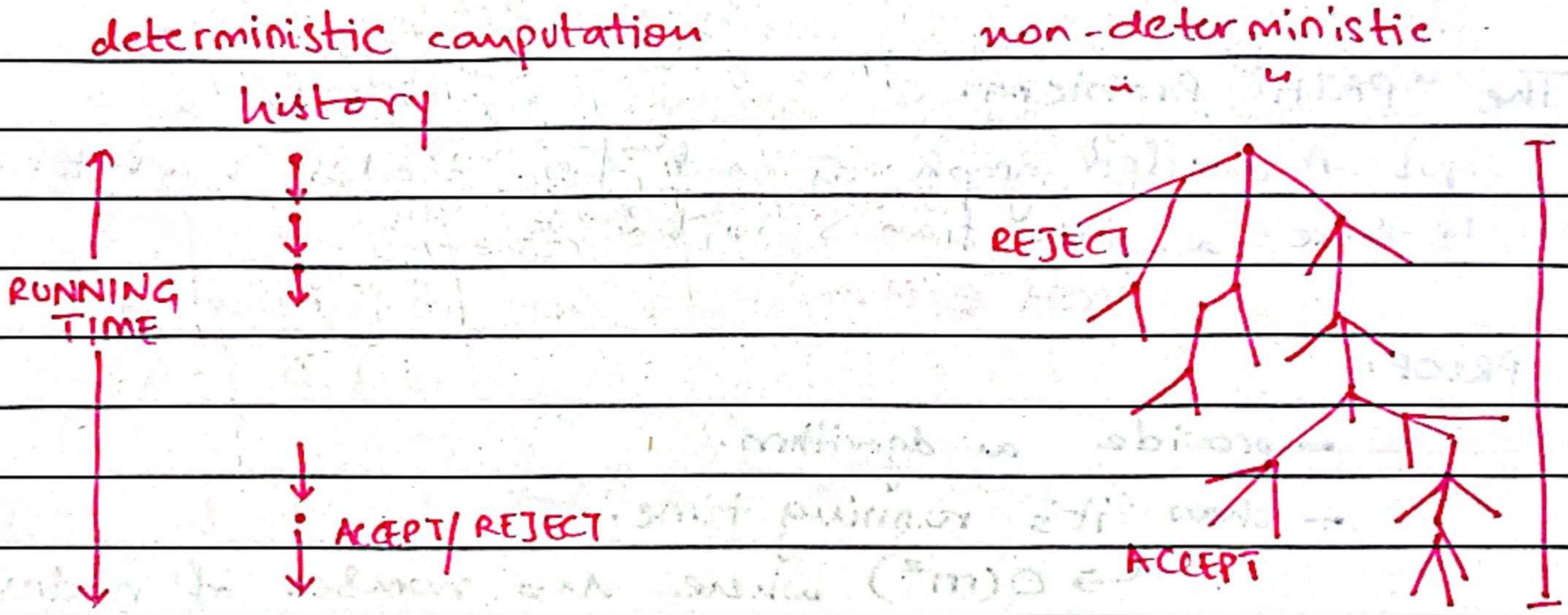
$t(n) \cdot O(t(n)) = O(t^2(n))$.

Date:_____

A polynomial-time algorithm will remain polynomial-time regardless of details of model of caputation.

NON-DETERMINISTIC T.M.S.

Running Time:
The number of steps the TM uses on the largest branch of caputation



every non deterministic TM can be simulated on a deterministic TM using EXPONENTIALLY MORE STEPS.

Date:_____

The Complexity classes P and NP.

The class P.

All reasonable deterministic models of computation are

POLYNOMIALLY EQUIVALENT

The class of languages that can be decided...
[ie the set of problems that can be solved...]
in POLYNOMIAL TIME on a Deterministic TM.

$$P = \bigcup_k \text{TIME}(n^k)$$

The "PATH" Problem.

Input: A directed graph G and two nodes s and t...
Is there a path from s to t?

PATH $\in$ P.

PROOF:

- provide an algorithm.
- show it's running time.
  - ↳ $O(n^2)$ where n = number of nodes.

Theorem:

Every context-free language is in P.

PROOF:

Provide an $O(n^3)$ algorithm A "Dynamic Programing" algorithm

Date:_____

→ use table to store partial results.
→ avoid having to recompute things over & over.
→ build bigger results out of smaller results
→ for 1 to N
→ compute all results of size i
→ store each result
→ make use of result.

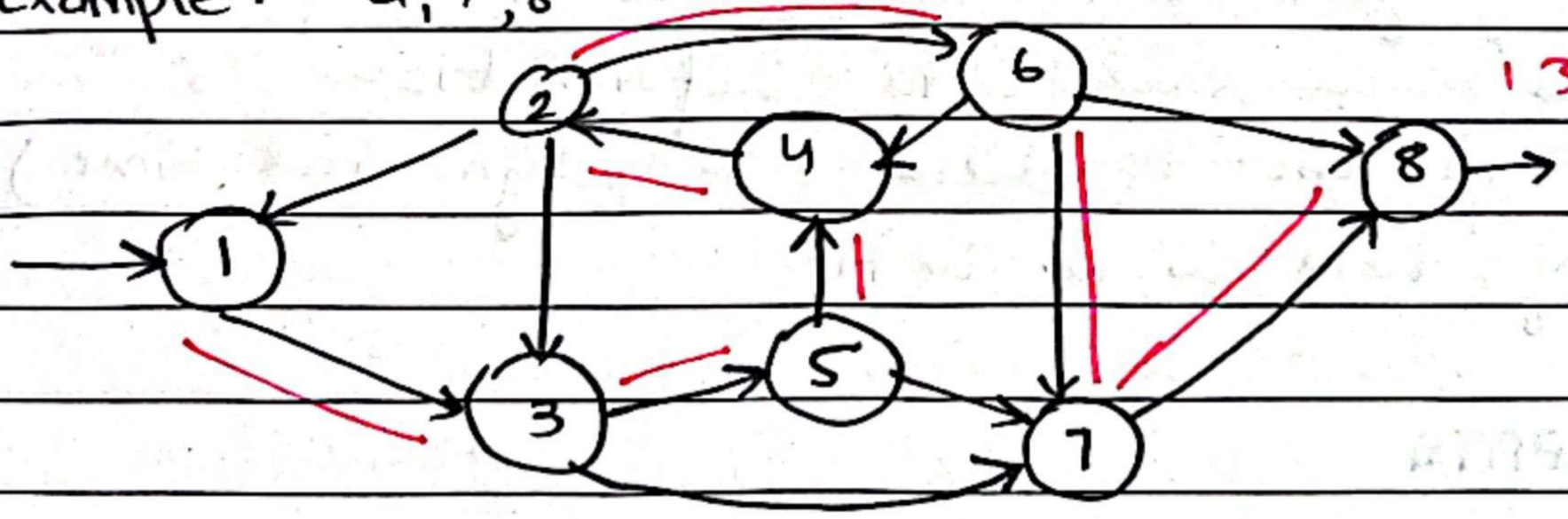
HAMPATH: The "Hamiltonian" Path Problem.

Given a directed graph, is there a path that goes through every node exactly once?

HAMPATH = { <G, s, t> | G is directed "HAMILTONIAN" graph & there is path from s to t }

we are given the starting and ending nodes.

Example: a, 1, 8



1 3 5 4 2 6 7 8

Date: _____

EXPONENTIAL PROBLEM

Generate all possible paths

1	2	3	4	5	6	7	8
1	4	3	2	8	7	5	6

⋮

↳ Test each path to see if it is legal.

NOTE This test can be done quickly in Polynomial Time.

This problem is in class NP.

It seems to require exponential time

But given the answer we can VERIFY it in polynomial time.

POLYNOMIAL VERIFIABILITY.

Given a language A a "VERIFIER" is an algorithm that is given some extra information "c" which it can use to check (in polynomial time) to verify that w is in A.

Example: HAMPATH

Given a problem such as.

$w = \langle G, s, t \rangle$.

is there a HAMILTONIAN PATH? EXPONENTIALLY HARD!

Date:_____

But verifier algorithm is passed some info
c = "1 3 5 4 2 6 7 8" and can confirm
that w ∈ HAMPATH
in POLYNOMIAL TIME

Definition:
A "verifier" for language A is an algorithm V
where
A = { w | V accepts <w, c> for some string c }
A "POLYNOMIAL-TIME VERIFIER" runs in polynomial time in n the legth of w.

A language is "POLYNOMIALY VERIFIABLE" if it has a polynomial-time verifier.

The string "c" is called "CERTIFICATE"

- Definition:
"NP" is class of languages that have polynomid verifiess.

Date: _____

A language is in NP iff it is decided by some NON-DETERMINISTIC POLYNOMIAL TIME Turing Machine.

↳ sometimes given as the definition of "NP"

PROOF

→ convert polynomial-time verifier into an equivalent polynomial-time non-deterministic Turing Machine.

PROOF

→ Input: w (length of n)

→ Algorithm

→ non-deterministically guess string c (length almost n^k)

→ run V on <w, c>

→ if V accepts, accept. Else, reject

→ Assume you have polynomial-time non-deterministic TM. construct a polynomial time verifier.

Date:_____

The verifier:

→ input <w,c>

→ algorithm

→ simulate the non-deterministic TM.

→ Use c as a guide about which choice to make at each step

→ If this branch accepts then ACCEPT, else REJECT

P = The class of languages for which membership can be decided quickly

NP = The class of languages for which membership can be verified quickly.

Definition

NTIME(t(n)) = { L | L is a language decided by an O(t(n)) time nondeterministic TM }

TIME(n^2) = The set of languages that can be decided by a DETERMINISTIC TM in $O(n^2)$ time.

NTIME(n^2) = The set of languages that can be decided by a NON-DETERMINISTIC TM in $O(n^2)$ time.

Date: _____

$$NP = \bigcup_{k} NTIME(n^k)$$

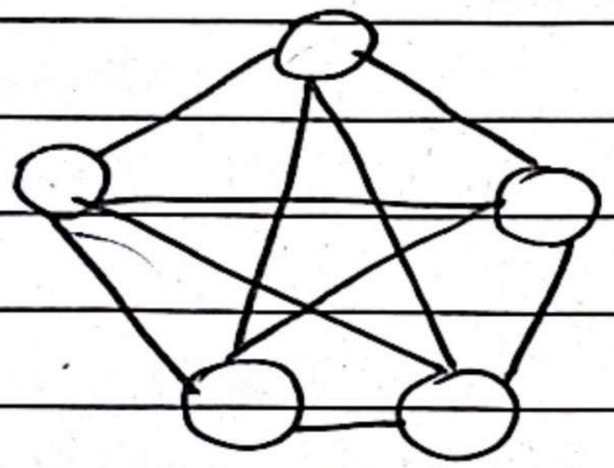
The "CLIQUE" Problem.

Given an undirected graph ...

A "clique" is a set of nodes such that every node in the clique is connected to every other node in clique.

A k—clique is a clique with k nodes.

* every node in k-clique should have k connections



5- Clique.

$$CLIQUE = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \}$$

Theorem: CLIQUE $\in$ NP.

PROOF

→ provide a polynomial-time verifier

— OR —

→ provide a polynomial-time non-deterministic TM.

Date:_____

The class "P"

The class of languages that can be decided...
in polynarial time of on a
DETERMISTIC TM.

The class "NP"

The class of languages that can be decided...
in polynarial time of a NONDETERMINISTIC
TURING.

P=NP } which is it?
P⊂NP }

There are lots of problems known to be in NP
These problems require EXPONENTIAL TIME to solve.

$$EXPTIME = \bigcup_k TIME(2^{n^k})$$

Results

$P \subseteq NP \subseteq EXPTIME$

Apparently, $P \subset NP = EXPTIME$

But this is also possible:
$P = NP \subset EXPTIME$.