Bittoo TRADE MARK REGD.

Date :
Page no.

Multitrack Turing Machine

a	a	b	B	....
b	c	b	B	....
c	b	c	B	....

↑ Finite control

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

$\Rightarrow \Sigma = \{\begin{matrix} a \\ b \\ c \end{matrix}, \begin{matrix} a \\ a \\ b \end{matrix} \cdots\}$

a b c	a c b	b b c	B B B	....

↑ Finite control

Single Track TM

Multitrack Turing Machine

$\delta(q_0, [a, b, c]) = (q_1, [b, b, b], L/R)$

* Can be convertable to single track TM.

Q. Design TM to multiply 2 integers

Sol^n:

$3 \times 2 = 2 + 2 + 2$

0
1
2

$3 \times 2 = 0 + 2 + 2 + 2$

$= 6$

Answer

b	b	b	b	0	0	0	1	0	0	1	b	b	b	b	b	b	

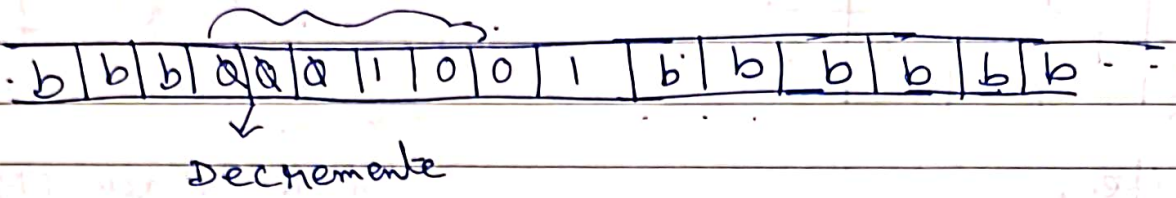
$m = 3$ $n = 2$

separator separator

The steps

1. $0^m 1 0^n 1$ is placed on the tape (the o/p will be written after the rightmost 1)
2. The leftmost '0' is erased
3. A block of n 0's is copied onto the right end. (concept in $3 \times 2 = 2 + 2 + 2$)

4. Step 2 & 3 are repeated m times and $1 0^m 1 0^{mn}$ is obtained on the tape.



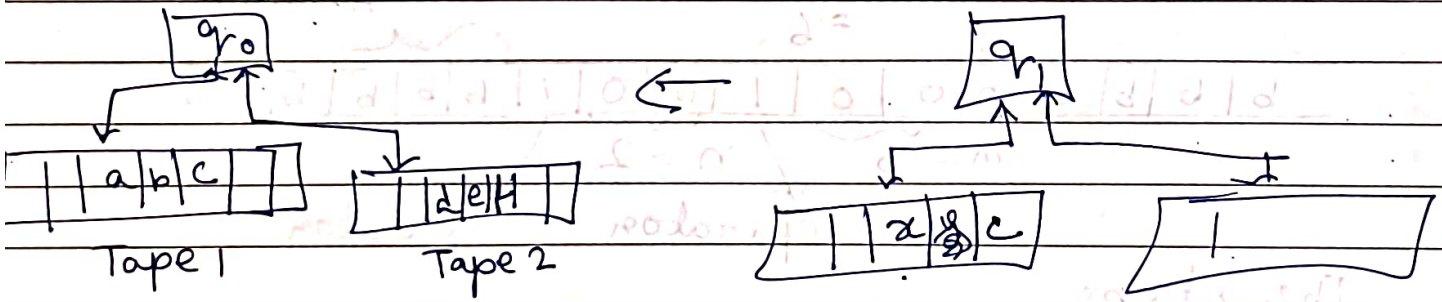
Subroutine

This will have a initial state and a return state. After reaching the return state, there is a ~~temporary~~ temporary halt.

Non-Deterministic Turing Machine

eg. if $n = 2$ with current configuration

$$\delta(q_0, a, e) = (q_1, x, y, L, R) \; N = 2$$



Bittoo

Date :
Page no.

Non - Deterministic Turing Machine

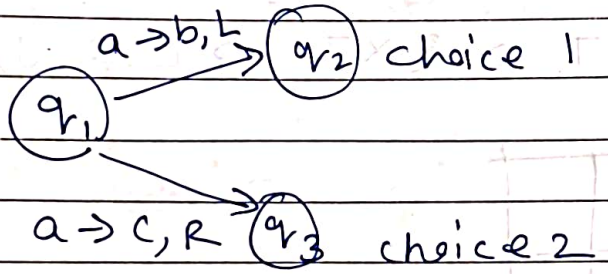
→ It is similar to DTM except that for any i/p example and current state it has a no. of choices.

→ A string is accepted by a NDTM if there is sequence of moves that heads to a final state.

→ The transition fns. (Multiple choices.)

$$\delta : Q \times X \rightarrow 2^{Q \times X \times (L,R)}$$

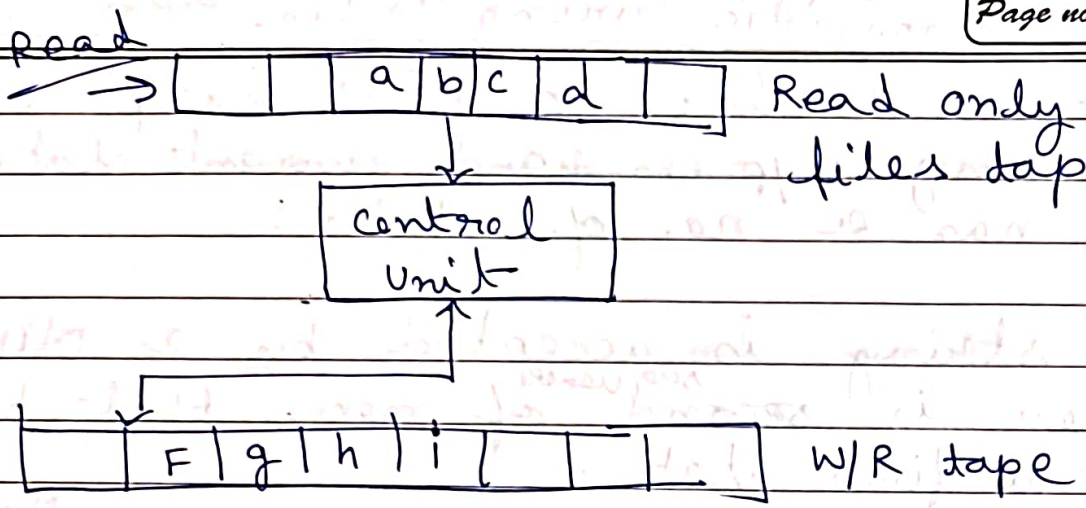
→ A ND TM is allowed to have more than one transition fn. a given tape symbol.



OFF line turing machine

Has 2 tapes.

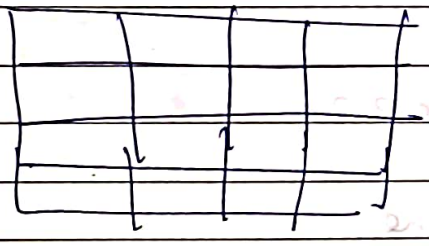
1. One tape is read only and contains the i/p.
2. The other is read - write and is initially blank

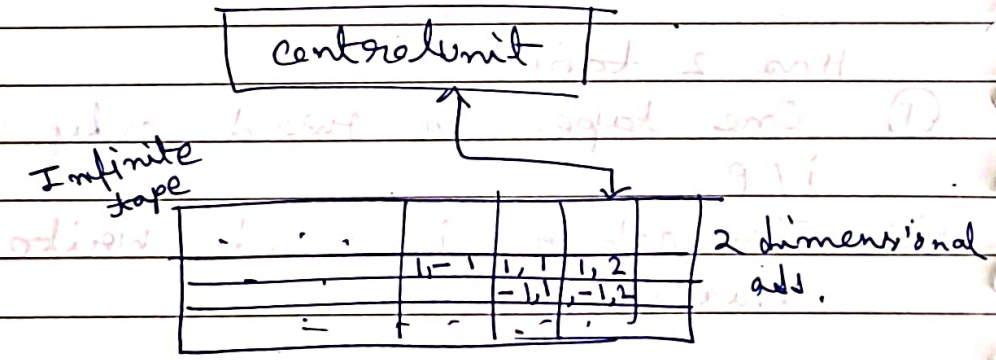


Multidimensional turing machine

→ It has a multidimensional tape eg. A two dimensional T.M. would read and write on an infinite tape divided into squares, like a checkboard

→ $\delta : Q \times T \rightarrow Q \times T \times \{L, R, U, D\}$





Date :
Page no.

Recursive & Recursively Enumerable Language :-

⇒ The T.M. may
- Halt & accept the i/p
- Halt & Reject the i/p, or
- Never halt / loop

Recursively Enumerable lang :-

There is a T.M for a language which accept every string otherwise not.

Recursive lang :-

There is a T.M. for a language which halt on every string.

Halting Problem :-

Halting problem is undecidability, It is not a problem, it just asks questions :- "is it possible to to tell whether a given m/c will halt for same given i/p".

eg. Input : A T.M. & i/p string w

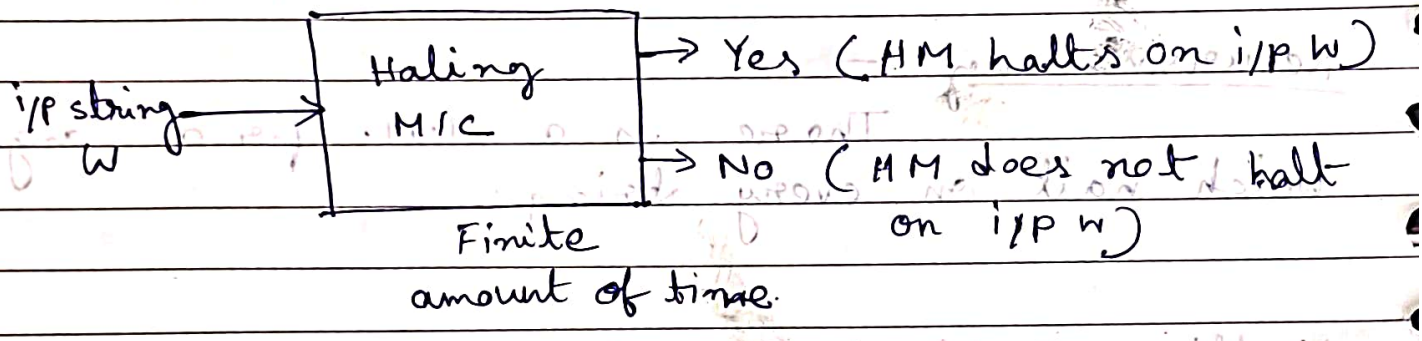
Problem : Does the TM finish computing of the ~~storing~~ string w in a finite no. of steps ? The answer must be yes or no.

Proof : Assume T.M. exists to solve this problem & then we will show it is contradicting itself.

We will call this TM as a Halting machine that produces a 'yes' or 'no' in a finite amount of time.

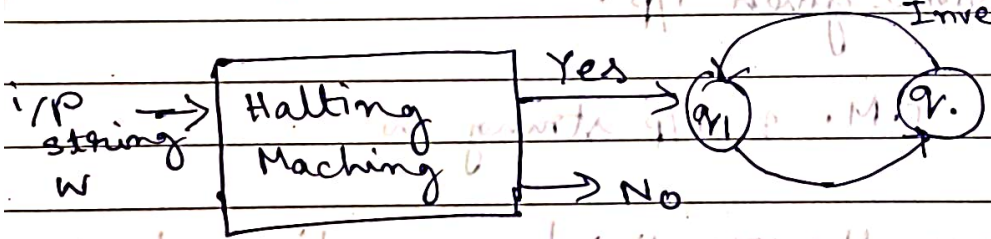
→ If the halting m/c finishes in a finite amount of time, and o/p comes as 'yes' otherwise 'no'.

The block diagram of Halting m/c :-



Now, we will design an inverted halting M/C as
— If H returns yes, then loop forever
— If H returns No, then Halt

The diagram for inverted halting m/c :-



Date :
Page no.

Complexity classes

→ Time complexity

- How long computation takes to execute
- In TM, this could be measured as no: of moves which are required to perform computation
- no. of m/c cycles.

→ Space complexity

- How much storage is required for computation
- In TM no. of cells are used
- No. of bytes used.

Types of complexity classes :

(1) P- class : Set of decision problem is solvable in polynomial time or in the class P.

If there exists an algorithm A such that A is an algo.

→ A takes instance of D as i/p

→ A always o/ps the connect answer "yes" or "no"

→ There exists a polynomial P such that the execution of A on the i/p of size n always terminates in p(n) or fewer steps.

eg. The minimum spanning tree problem is in class P.

The class P is often considered as synonymous with the class of computationally feasible problems, although in practice this is somewhat unrealistic.

NP class :—

A decision problem is non deterministically polynomial time solvable or in the class NP if there exists an Algorithm A such that,

→ There exists a polynomial p such that for each potential witnesses of each instances of size of D. The execution of the algorithm A takes at most $p(n)$ steps.

→ Think of a non-deterministic computer as a computer that magically 'guesses' a solution, then has to verify that it is correct.

- If solution exists, computer always guess it.
- One way to imagine it : a parallel computer that can freely spawn an infinite no. of processes.

Date :
Page no.

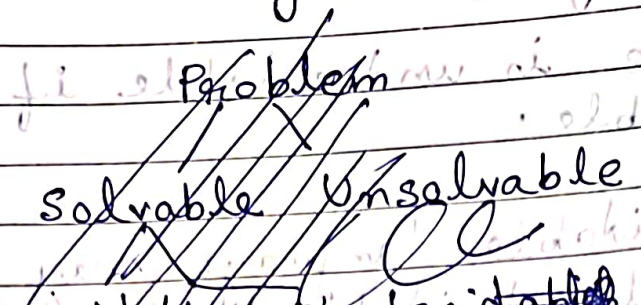
- All processors attempts to verify that their sol. works.

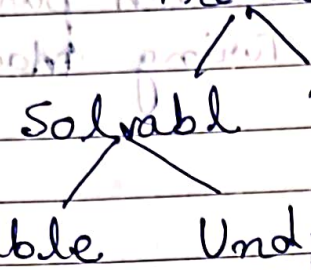
- If a processor finds it has a working sol^{n} o

so. NP = Problem verifiable in polynomial time

→ Every problem in this class can be solvable in exponential time using exhaustive search.

~~Decidability~~ Decidability





Decidability & Undecidability :

Decidable Languages :-

A language 'L' is decidable if it is a recursive language. All decidable languages are recursive language are recursive languages and vice-versa

Partially Decidable language :-

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

Undecidable language :—

→ A language is undecidable if it is not decidable.

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

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

Recursive Language	TM will always Halt
Recursively Enumerable Language	TM will halt sometime & may not halt sometimes
Decidable Language	Recursive Language
Partially Decidable Language	Recursively Enumerable Language
UNDECIDABLE	No. TM for that lang.

Bittoo TRADE MARK REGD.

Date :
Page no.

DECIDABILITY Overview

→ Every question about regular languages is decidable.

→ Some questions about CFLs are decidable, but some are not.

→ The "Halting Problem" is not decidable.

→ Some languages are not turing recognizable.

→ Many questions about turing machines are not ~~decid~~ decidable and some are not even turing recognizable.

Q: Given a DFA and a string, will the DFA accept?
Is this problem decidable?

Sol^n: Languages are decidable. We must express the problem in terms of languages.

Theorem: The language:-

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

↙ Acceptance

..... is decidable

$\langle\ \rangle$ → Denotes something abstract value/coding

Possible Confusion:

1. String w
   language $DFA = RL$ ← Regular Lang.

2. String $\langle B, w\rangle$
   &
   Language A_{DFA} ← Not Regular, Not CFG, But it is decidable.

Proof:

→ Provide a TM that decides it.
→ The TM is given as i/p $\langle B, w\rangle$
→ A DFA and a string w.
→ The TM checks to make sure that B is a valid representation.
→ The TM then simulates B on w.
→ If B reaches A final state at the end of w.
→ Otherwise the TM will reject.
→ This TM will always halt.

Bittoo

Date :
Page no.

Cook's Theorem :

The satisfiability problem (SAT) is NP comple.
or
L

SAT ?

A propositional logic formula ϕ is called satisfiable if there is some assignment to its variable that makes it evaluate to true.

→ $P \wedge q$ is satisfiable $P=1, q=1$
P or q

→ $P \wedge \neg P$

| Operator : NOT ¬ (highest
AND ·
OR + |
|---|

3SAT

A language 3SAT = $\{\phi 1, \phi$

Bittoo TRADE MARK REGD.

Date :
Page no.

Undecidability of the Halting Problem

Given a program, will it 'halt'?

Can we design a maching which if given a program can find out or decide if that program will always halt or not, halt on a particular i/p?

Let us assume that we can:

H (P, I) ← H can take 2 arguments.
- → Halt.
- → Not Halt

This allows us to write another program:

```
C (X)
{ if { H (X, X) == Halt }
      Loop forever;
  else
      Return;
}
```

If we run 'C' on itself

C (C)

H (C, C) == Halt → Not Halt

H (C, C) == Not Halt → Halt

This means we cannot actually design a ~~maching~~ machines which can say us that a machine can tells us a program can halt or not.

Bittoo

Date :
Page no.

Complexity:

* When a problem / language is decidable, it simply means that the problem is computationally solvable in principle.

Growth Rate of Functions:

When we have 2 algorithms for the same problem, we may require a comparison between the running time of these 2 algorithms..

Definition: Let $f, g : N \to R^+$ (R^+ being the set of all ~~possible~~ +ve real nos.) we say that $f(n) = O(g(n))$ if there exist +ve integers c and N_0 such that

$$f(n) \leq c\,g(n) \text{ for all } n \geq N_0$$

Here, f is 'big Oh' of g.

** $f(n) = O(g(n))$ is not an eq^n, it only ~~denotes~~ expresses a relation btn. 2 fns. f and g.

Q: Let $f(n) = 4n^3 + 5n^2 + 7n + 3$.
P.T. $f(n) = O(n^3)$

Sol^n: Let us take ~~not acceptable for this Prob.~~ $c = 5$ and $N_0 = 10$

$\therefore f(n) = 4n^3 + 5n^2 + 7n + 3$ for $n \geq 10$

$\leq cn^3$

take $n = 10$; $f(n) = 573 < 10^3$ $\therefore f(n) = O(n^3)$

$n > 10$ e.g. $n = 11$ $< 11^3$

$< n^3$

Bittoo

Date :

Page no.

Theorem:

If $P(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ is a polynomial degree k over $\mathbb{Z}$ and $a_k > 0$, then $p(n) = O(n^k)$

Proof: $P(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$

$\because a_k$ is +ve int. & $a_k \geq 1$

Let us assume N_0 such that $n \geq N_0$

$$\therefore \frac{|a_{k-1}|}{n}, \frac{|a_{k-2}|}{n^2}, \ldots, \frac{|a_1|}{n^{k-1}}, \frac{|a_0|}{n^k} \leq \frac{1}{k}$$

$$\therefore \left| \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \cdots + \frac{a_0}{n^k} \right| < 1$$

As $a_k \geq 1$, $\frac{P(n)}{n^k} = a_k + \frac{a_{k-1}}{n} + \cdots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k} > 0$ for all $n \geq N_0$

Also

$$\frac{P(n)}{n^k} = a_k + \left(\frac{a_{k-1}}{n} + \cdots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k} \right)$$

$$\leq a_k + 1$$

So

The classes P and NP

P - class

→ A TM is said to be of time complexity $T(n)$ if the following holds: Given an input w of length n, M halts after making at most $T(n)$ moves.

→ A language L is in class P if there exists some polynomial $T(n)$ such that $L = T(M)$ for some deterministic TM M of time complexity $T(n)$.

~~construct the time complexity T(n) for the TM, as given~~

Q: Find the running time for the Euclidean Algo. for evaluating gcd(a,b) where a & b are +ve integers expressed in binary representation.

Sol^n

⇐ Polynomial algorithm

The Euclidean algo. is based on the below facts

1. If we subtract smaller number from larger no., GCD doesn't change. so, if we keep subtracting repeatatedly the larger of 2, we end up with GCD.

2. Instead of subtraction, if we dive smaller no., the algorithm stops when we find remainder '0'.

The classes P and NP

P - class

→ A TM is said to be of time complexity $T(n)$ if the following holds: Given an input w of length n, M halts after making at most $T(n)$ moves.

→ A language L is in class P if there exists some polynomial $T(n)$ such that $L = T(M)$ for some deterministic TM M of time complexity $T(n)$.

~~construct the time complexity T(n) for the TM's given~~

Q: Find the running time for the Euclidean Algo. for evaluating gcd(a,b) where a & b are +ve integers expressed in binary representation.

Sol^n: ← Polynomial algorithm

The Euclidean algo. is based on the below facts

1. If we subtract smaller number from larger no., GCD doesn't change. so, if we keep subtracting repeatatedly the larger of 2, we end up with GCD.

2. Instead of subtraction, if we dive smaller no., the algorithm stops when we find remainder '0'.

e.g. 36, 60

$36 = 2 \times 2 \times 3 \times 3$

$60 = 2 \times 2 \times 3 \times 5$

GCD = Multiplication of common factors

$= 2 \times 2 \times 3$

$= 12$

In other words:

⇒ The algo has the following steps:

1. The i/p is (a, b)
2. Repeat unit b = 0
3. Assign a ← a mod b
4. Exchange a & b
5. Output a

step 3 replaces a by a mod b.

If $a/2 \geq b$, then $a \bmod b < b \leq a/2$.

If $a/2 < b$,

then $a < 2b$

Write $a = b + r$ for some $r < b$

Then $a \% b = r < b < a/2$

∴ $a \% b < a/2$

So, a is reduced by at least half in size on the application of step 3.

∴ One iteration of step 3 & 4 reduces a & b by at least half in size.

Bittoo
Date :
Page no.

∴ The max^{m} no. of execution of step 3 & 4 are $(\log_2 a)$ and $(\log_2 b)$

The no. of iterations of step 3 & 4 is $O(n)$.

⇒ we have to perform step 2 at most $\min\{[\log_2 a], [\log_2 b]\}$ times or n times.

∴ $T(n) = n\,O(n) = O(n^2)$

NP Class :

⇒ A language L is in class NP if there is a non-deterministic TM, M and a polynomial time complexity $T(n)$ such that $L = T(M)$ and M executes at most $T(n)$ moves for every input w of length n.

Polynomial time Reduction & NP completeness:

Theorem :

- If there is a polynomial time reduction from P_1 to P_2 and if P_2 is in P, then P_1 is also in P.

2. Let L be a language or problem in NP. Then L is NP-complete if

i) L is in NP
ii) For every language L' in NP there exists a polynomial-time reduction of L' to L.
iii) The class of NP complete language is a subclass of NP.

3. If P_1 is NP-complete and there is a polynomial time reduction of P_1 and P_2, then P_2 is NP-complete.

4. If some NP-complete problem is in P, the P = NP.

Proof L is any lang. in NP.
*** We will show that there is a polynomial time reduction of L to P_2.

As; P_1 is NP-complete
so, time reduction of L to P_1

suppose n - size i/p string W
& so time taken to convert this
x → P_1
y → P_2

∴ Time taken for transforming w to y is at most $P_1(n) + P_2(P_1(n))$

As $P_1(n) + P_2(P_1(n))$ is a polynomial.

∴ we get a polynomial-time reduction of L to P_2.

∴ P_2 is NP-complete.

Time taken for transforming x to y is at most $P_1(n) + P_2(P_1(n))$

As $P_1(n) + P_2(P_1(n))$ is a polynomial.

∴ we get a polynomial-time reduction of L to P_2.

∴ P_2 is NP-complete.