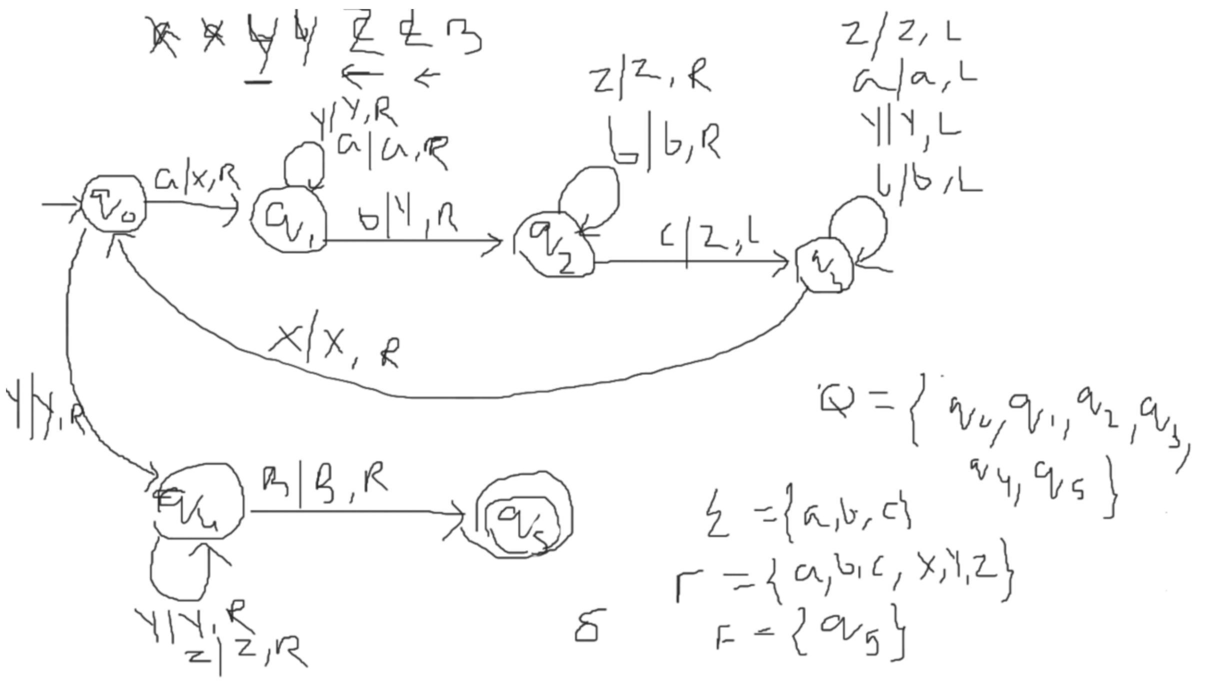
# IAT 3 Solution

## Sub: Automata Theory and Computability(18CS54)

## Dept. of CSE, CMRIT, Bangalore

**Q. 1. Design a Turing Machine for L= {$a^n b^n c^n$|n>=1}. Write the transition function for the same and also indicate the moves made by TM for input string W=aabbcc**

**Ans:**

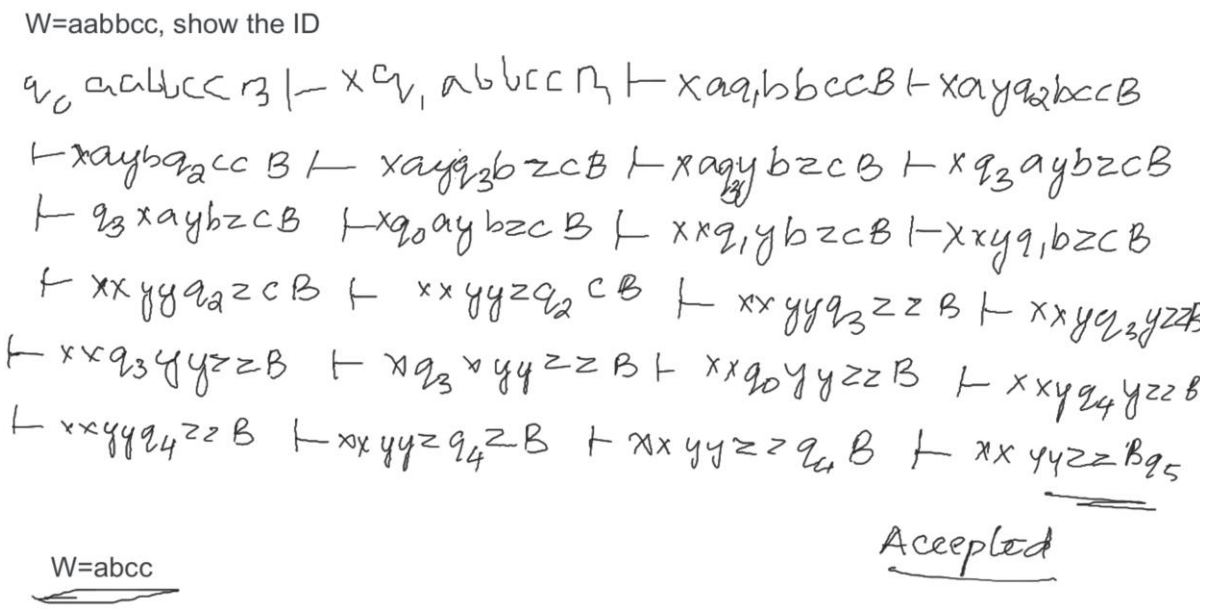


$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, X, Y, Z\}$

$F = \{q_5\}$



W=aabbcc, show the ID

$q_0 aabbccB \vdash xq_1abbccB \vdash xaq_1bbccB \vdash xayq_2bccB$

$\vdash xaybq_2ccB \vdash xayq_3bzcB \vdash xaq_3ybzcB \vdash xq_3aybzcB$

$\vdash q_3xaybzcB \vdash xq_0aybzcB \vdash xxq_1ybzcB \vdash xxyq_1bzcB$

$\vdash xxyyq_2zcB \vdash xxyyzq_2cB \vdash xxyyq_3zzB \vdash xxyq_3yzzB$

$\vdash xxq_3yyzzB \vdash xq_3xyyzzB \vdash xxq_0yyzzB \vdash xxyq_4yzzB$

$\vdash xxyyq_4zzB \vdash xxyyzq_4zB \vdash xxyyzzq_4B \vdash xxyyzzBq_5$

Aceepted

W=abcc

**Q.2. Explain the working principle of TM with diagram. Explain variants of TM with a neat diagram.**

**Ans:**

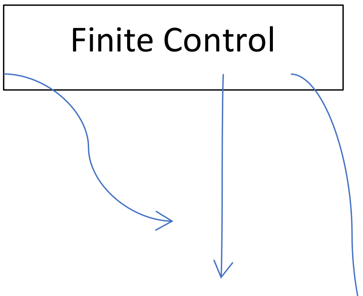
**VARIANTS OF TURING MACHINE:**

There are two new models of Turing machines:

1. MULTITAPE TURING MACHINE
2. NON-DETERMINISTIC TURING MACHINE

# MULTITAPE TURING MACHINE

Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head. Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank. At first, the first tape is occupied by the input and the other tapes are kept blank. Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.



| | | | | | |
|---|---|---|---|---|---|
| . . . | | | | | . . . |

| | | | | | |
|---|---|---|---|---|---|
| . . . | | | | | . . . |

| | | | | | |
|---|---|---|---|---|---|
| . . . | | | | | . . . |

A Multi-tape Turing machine can be formally described as a 7-tuple ($Q,\Sigma,\Gamma, B, \delta, q_0, F$) where −

- **Q** is a finite set of states
- Σ is a finite set of inputs
- Γ is the tape alphabet
- **B** is the blank symbol
- **δ** is a relation on states and symbols where

  $\delta: Q \times \Gamma^k \rightarrow Q \times (\Gamma \times \{\text{Left, Right, Stationary}\})^k$

  where there is **k** number of tapes
- **$q_0$** is the initial state
- **F** is the set of final states

  In each move the machine M:

  (i) Enters a new state

  (ii) A new symbol is written in the cell under the head on each tape

  (iii) Each tape head moves either to the left or right or remains stationary.

# NON-DETERMINISTIC TURING MACHINE

In a Non-Deterministic Turing Machine, **for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic.** The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

**An input is accepted if there is at least one node of the tree which is an accept configuration**, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

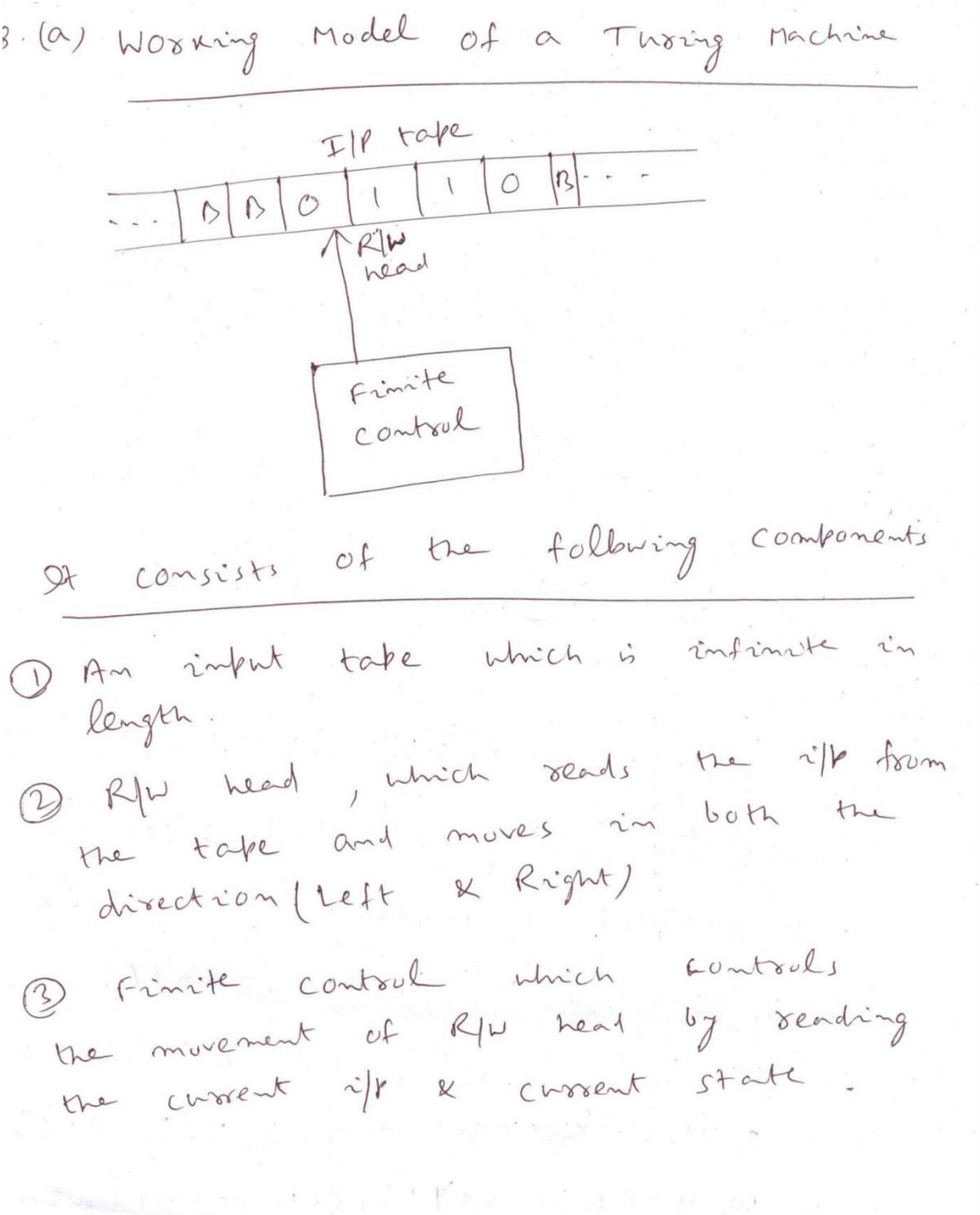
A non-deterministic Turing machine can be formally defined as a 7-tuple (Q, ∑,Γ, δ, $q_0$, B, F) where −

- **Q** is a finite set of states
- Γ is the tape alphabet
- ∑ is the input alphabet
- **δ** is a transition function;

  $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{\text{Left, Right}\})}$

- **$q_0$** is the initial state
- **B** is the blank symbol
- **F** is the set of final states

**Q**



3. (a) Working Model of a Turing Machine

I/P tape
. . . B B 0 1 1 0 B . . .
R/W head
Finite control

It consists of the following components

① An input tape which is infinite in length.

② R/W head, which reads the i/p from the tape and moves in both the direction (Left & Right)

③ Finite control which controls the movement of R/W head by reading the current i/p & current state.

**Q.3. Write Short notes on:**

**(a) Halting problem of TM**

**(b) Classes of P and NP**

# Halting problem of TM

VTU : Dec.-16, Marks 5

To state halting problem we will consider the given configuration of a turing machine. The output of TM can be

i) **Halt :** The machine starting at this configuration will halt after a finite number of states.

ii) **No Halt :** The machine starting at this configuration never reaches a halt state, no matter how long it runs.

- Now the question arises based on these two observation : Given any functional matrix, input data tape and initial configuration, then is it possible to determine whether the process will ever halt? This is called **halting problem.**
- That means we are asking for a procedure which enable us to solve the halting problem for every pair (machine, tape). The answer is "no".
- That is the halting problem is unsolvable.
- Now we will prove why it is unsolvable.
- Let, there exists a TM $M_1$ which decides whether or not any computation by a TM T will ever halt when a description $d_T$ of T and tape t of T is given [That means the input to machine $M_1$ will be (machine, tape) pair].
- Then for every input (t, $d_T$) to $M_1$ if T halt for input t, $M_1$ also halts which is called **accept halt.**
- Similarly if T does not halt for input t then the $M_1$ will halt which is called **reject halt.** This is shown in Fig. 7.4.1

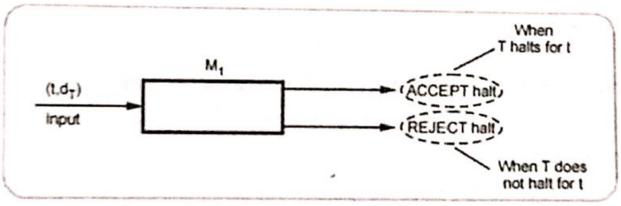


**Fig. 7.4.1**

- Now we will consider another Turing Machine $M_2$ which takes an input $d_T$.
- It first copies $d_T$ and duplicates $d_T$ on its tape and then this duplicated tape information is given as input to machine $M_1$. But machine $M_1$ is a modified machine with the modification that whenever $M_1$ is supposed to reach an accept halt, $M_2$ loops forever.

- Hence behavior of $M_2$ is as given. It loops if T halts for input t = $d_T$ and halts if T does not halt for T = $d_T$.
- The T is any orbitary Turing machine.

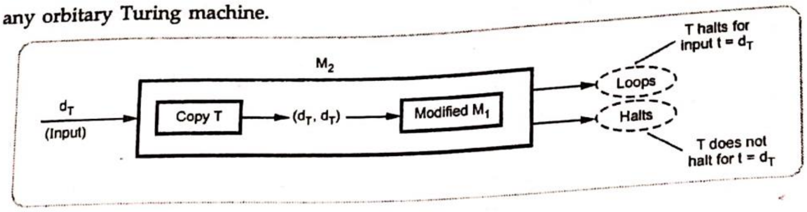


**Fig. 7.4.2**

As $M_2$ itself is one turing machine we will take $M_2$ = T. That means we will replace T by $M_2$ from above given machine.

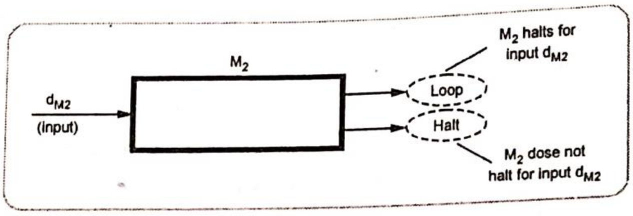
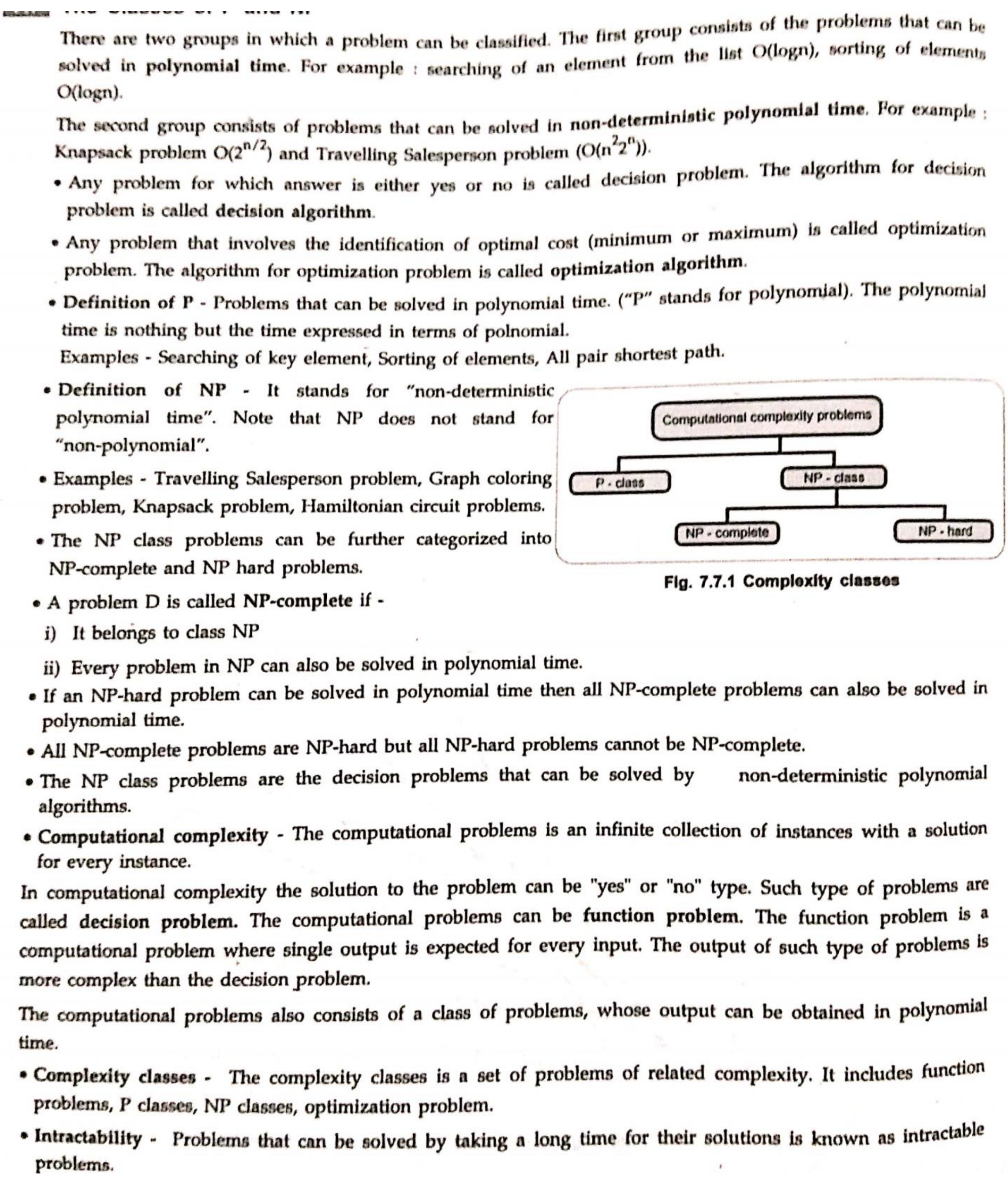


**Fig. 7.4.3**

Thus machine $M_2$ halts for input $d_{M2}$ if $M_2$ does not halt for input $d_{M2}$. This is a contradiction. That means a machine $M_1$ which can tell whether any other TM will halt on particular input does not exist. Hence halting problem is unsolvable.

# Classes of P and NP



There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in **polynomial time**. For example : searching of an element from the list O(logn), sorting of elements O(logn).

The second group consists of problems that can be solved in **non-deterministic polynomial time**. For example : Knapsack problem $O(2^{n/2})$ and Travelling Salesperson problem $(O(n^2 2^n))$.

- Any problem for which answer is either yes or no is called decision problem. The algorithm for decision problem is called **decision algorithm**.
- Any problem that involves the identification of optimal cost (minimum or maximum) is called optimization problem. The algorithm for optimization problem is called **optimization algorithm**.
- **Definition of P** - Problems that can be solved in polynomial time. ("P" stands for polynomial). The polynomial time is nothing but the time expressed in terms of polnomial.
  Examples - Searching of key element, Sorting of elements, All pair shortest path.
- **Definition of NP** - It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".
- Examples - Travelling Salesperson problem, Graph coloring problem, Knapsack problem, Hamiltonian circuit problems.
- The NP class problems can be further categorized into NP-complete and NP hard problems.

**Fig. 7.7.1 Complexity classes**

- A problem D is called **NP-complete** if -
  i) It belongs to class NP
  ii) Every problem in NP can also be solved in polynomial time.
- If an NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.
- All NP-complete problems are NP-hard but all NP-hard problems cannot be NP-complete.
- The NP class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.
- **Computational complexity** - The computational problems is an infinite collection of instances with a solution for every instance.

In computational complexity the solution to the problem can be "yes" or "no" type. Such type of problems are called **decision problem.** The computational problems can be **function problem.** The function problem is a computational problem where single output is expected for every input. The output of such type of problems is more complex than the decision problem.

The computational problems also consists of a class of problems, whose output can be obtained in polynomial time.

- **Complexity classes** - The complexity classes is a set of problems of related complexity. It includes function problems, P classes, NP classes, optimization problem.
- **Intractability** - Problems that can be solved by taking a long time for their solutions is known as intractable problems.

**Q.4. Define Chomsky Normal Form (CNF). Convert the following CFG to CNF.**

**S→ aAa | bBb | ϵ**
**A→C | a**
**B→C | b**
**C→CDE | ϵ**
**D→A| B |ab**

**Ans:**

CNF stands for Chomsky normal form. A CFG (context free grammar) is in CNF (Chomsky normal form) if all production rules satisfy one of the following conditions:

- A non-terminal generating two non-terminals. For example, S → AB.
- A non-terminal generating a terminal. For example, S → a.

**Step 1: Remove NULL productions**

**NULL** set= { S, A, B, C, D}

**S→ aAa | bBb | aa | bb**
**A→C | a**
**B→C | b**
**C→CDE | DE | CE | E**
**D→A| B |ab**

**Step2: Remove useless symbol**

**C, E, D are useless symbols.**

**S→ aAa | bBb | aa | bb**
**A→ a**
**B→ b**

**No Unit Productions.**

**Step4: Convert to CNF**

**S→ PX | QY | XX | YY**
**A→ a**
**B→ b**
**X→a**
**Y→b**
**P→XA**
**Q→YB**

**Q.5. Write Short notes on:**
**Post correspondence problem**

**Post Correspondence Problem** is a popular undecidable problem that was introduced by Emil Leon Post in 1946. It is simpler than Halting Problem.
In this problem we have N number of **Dominos** (tiles). The aim is to arrange tiles in such order that string made by Numerators is same as string made by

Denominators.
In simple words, let’s assume we have two lists both containing N words, aim is to find out concatenation of these words in some sequence such that both lists yield same result.
Let’s try understanding this by taking two **lists A** and **B**

`A= [aa, bb, abb] and B=[aab, ba, b]`

Now for sequence 1, 2, 1, 3 first list will yield aabbaaabb and second list will yield same string aabbaaabb.
So the solution to this PCP becomes 1, 2, 1, 3.

## Linear Bounded Automata

Alinear bounded automata (**LBA**) is a restricted form of Turing machinewith a tape of some bounded finite length. The computation is restricted to the constant bounded area. The input alphabet contains two special symbols which serve as left end markers and right end markers which mean **the transitions neither move to the left of the left end marker nor to the right of the right end marker of the tape.**

A linear bounded automaton can be defined as an 8-tuple (Q, ∑,Γ, $q_0$, $M_L$, $M_R$, δ, F) where −

- **Q** is a finite set of states
- Γ is the tape alphabet
- ∑ is the input alphabet
- **$q_0$** is the initial state
- **$M_L$** is the left end marker (ex: <)
- **$M_R$** is the right end marker(ex: >) where $M_R \neq M_L$
- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, Constant ‘c’) where c can be 0 or +1 or -1
- **F** is the set of final states

Working space

| | | | |
|---|---|---|---|
| < | a | b | > |

Left end marker Input string Right end marker

A deterministic linear bounded automaton is always **context-sensitive** and the linear bounded automaton with empty language is **undecidable.**

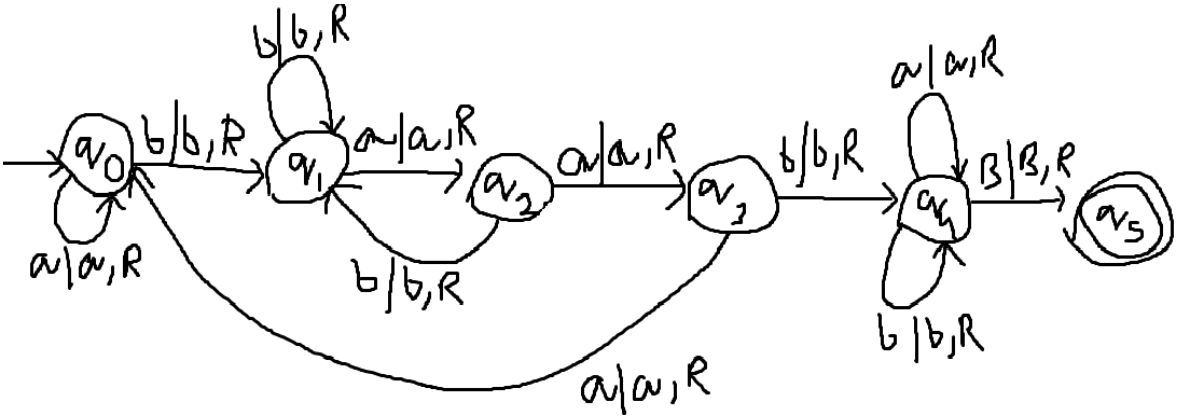
- The language accepted by LBA is called **context-sensitive language.**
- **Example:**

  $L=\{a^n b^n c^n\}$ and $L=\{ a^{n!} \}$

- It is more powerful that NPDA but less powerful that TM

**Q.6. Design a Turing Machine for L= {W contains a substring baab and W Ɛ {a,b}*}. Write the transition function for the same and also indicate the moves made by TM for input string W=abaabb.**

**Ans:**



Transition Function:

**δ (q0,a)= (q0,a,R)**
**δ (q0,b)= (q1,b,R)**

**δ (q1,a)= (q2,a,R)**
**δ (q1,b)= (q1,b,R)**

**δ (q2,a)= (q3,a,R)**
**δ (q2,b)= (q1,b,R)**

**δ (q3,a)= (q0,a,R)**
**δ (q3,b)= (q4,b,R)**

**δ (q4,a)= (q4,a,R)**
**δ (q4,b)= (q4,b,R)**

**δ (q4,B)= (q5,B,R)**

**ID for input string W=abaabb**

**(q0, abaabbB) |-- aq0baabbB |--abq1aabbB |--abaq2abbB |--abaaq3bbB |--abaabq4bB |-- abaabbq4B |--abaabbBq5 Accepted**

**Q.7. Define Greibach Normal Form (GNF). Convert the following CFG to GNF.**
**S→AA|0**
**A→SS| 1**

**Ans:**

GNF stands for Greibach normal form. A CFG (context free grammar) is in GNF (Greibach normal form) if all the production rules satisfy one of the following conditions:

- A non-terminal generating a terminal. For example, A → a.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, S → aASB.

Step1: No null production, no unit production, no useless symbols.

Step2:

$A_1 \rightarrow A_2A_2 \mid 0$

$A_2 \rightarrow A_1A_1 \mid 1$

In the $A_2$ production 2>1, so

$A_2 \rightarrow A_2A_2A_1 \mid 0A_1 \mid 1$

Removing left recursion from the grammar.

$A_1 \rightarrow A_2A_2 \mid 0$

$A_2 \rightarrow 0A_1 A_2' \mid 1 A_2' \mid 0A_1 \mid 1$

$A_2' \rightarrow A_2A_1 A_2' \mid A_2A_1$

Converting to GNF

$A_1 \rightarrow 0A_1 A_2' A_2 \mid 0 \mid 1 A_2' A_2 \mid 0A_1 A_2 \mid 1 A_2$

$A_2 \rightarrow 0A_1 A_2' \mid 1 A_2' \mid 0A_1 \mid 1$

$A_2' \rightarrow 0A_1 A_2' A_1 A_2' \mid 0A_1 A_2' A_1 \mid 1 A_2' A_1 A_2' \mid 1 A_2' A_1 \mid 0A_1A_1 A_2' \mid 0A_1A_1 \mid 1A_1 A_2' \mid 1A_1$