Applications of Automata Theory:

Finite Automata:

① String Matching / Processing

② Compiler Construction

The various compilers such as C/C++, Pascal, FORTRAN, or any other compiler is designed using the finite automata. The DFA's are extensively used in the building the various phases of compilers such as:

① Lexical Analysis: To identify the tokens, identifiers, to strip of the comments etc.

② Syntax Analysis: To check the syntax of each statement or control statement used in the program

③ Code optimization: To remove the unwanted code

④ Code generation: To generate the machine code.

The concept of finite automata is used in wide applications:

1. Large natural vocabularies can be described using finite automaton which includes the applications such as spelling checkers, and advisers, multi-language dictionaries, to redent the documents; in calculators to evaluate complex expressions based on the priority of an operator etc. to name a few. Any editor that we use uses finite automaton for implementation.

2. Finite automaton is very useful for recognizing difficult problems i.e sometimes it is very essential to solve an un-decidable problem. Even though there is no general soln exists for the specified problem, using TOC, we can find the approximate solns.

3. Finite automaton is very useful for h/w design such as circuit verification, for design of the h/w board (mother board or any other h/w unit), automatic traffic signals, radio controlled toys, elevators, automatic sensors, remote sensing or controller etc.

4. In game theory & games wherein we use some control characters to fight against a monster, computer graphics, linguistics etc., finite automaton plays a very important role.

* An Automaton is an abstract model of a digital computer. An abstract machine (also called abstract computer) is a conceptual or theoretical model of a computer hardware or s/w s/m which really does not exist. These machines are not actual machines and hence, they are also called hypothetical computers. These machines have commonly encountered h/w features and concepts and avoids most of the details that are often found in real machines. The various types of abstract machines are:

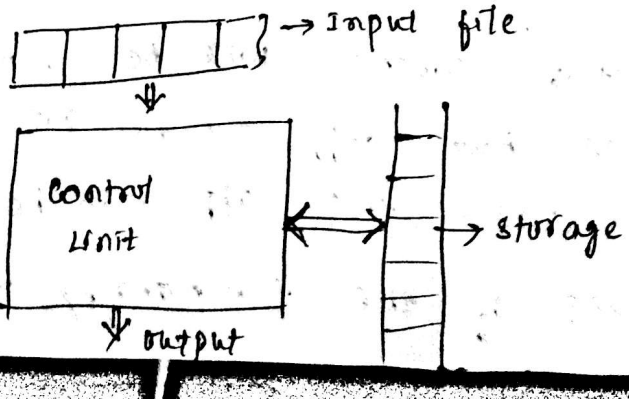
1. Finite Automata
2. Linear Bounded Automata
3. Push Down Automata
4. Turing Machine.

Every automaton includes some essential features. It has a mechanism for reading input. It will be assumed that the I/P is a string over a given alphabet, written on an I/p file, which the automaton can read but not change. The I/p file is divided into cells, each of which can hold one symbol. The I/p mechanism can read the I/p file from left to right, one symbol at a time. The I/p mechanism can also detect the end of the I/p string.

The automaton can produce o/p of some form. It may have a temporary storage device, consisting of an unlimited no of cells, each capable of holding a single symbol from an alphabet. The automaton can read and change the contents of the storage cells.

Finally, the automaton has a control unit, which can be in any one of a finite no of internal states & which can change state in some defined manner. i.e

ATM Example



An automaton whose output response is limited to a simple yes or no is called an accepter. Presented with an i/p string, an accepter either accepts the string or rejects it.

A more general automaton, capable of producing strings of symbols as outputs, is called a transducer.

An automaton is assumed to operate in a discrete time frame. At any given time, the control unit is in some internal state, & the i/p mechanism is scanning a particular symbol on the i/p file. The internal state of the control unit at the next time step is determined by the next state or transition function. This transition fn. gives the next state in terms of the current state, the current i/p symbol, & the info currently in the temporary storage. During the transition from one time interval to the next, o/p may be produced & the info in the temporary storage changed.

The term configuration will be used to refer to a particular state of the control unit, i/p file, & temporary storage.

The transition of the automaton from one configuration to the next will be called a move.

*. A finite-state control will be common to all specific cases, but differences will arise from the way in which the o/p can be produced & the nature of the temporary storage.

The nature of the temporary storage has the stronger effect on particular types of automata.

* Refer to diff. kinds of Automata. [page no 3 ppt].

* Introduction to Finite Automata:

Applications:

1) Formal languages & grammars are used widely in connection with prog. languages.

ie If we write a compiler, or if we wish to reason about the correctness of a program, a precise description of the language is needed at almost every step. Among the ways in which prog. languages can be defined precisely, grammars are perhaps the most widely used. [syntax diagrams]

2) Prog. languages can be defined precisely through grammars, & both grammars & automata play a fundamental role in the decision processes by which a specific piece of code is accepted as satisfying the conditions of a prog. languages.

3) Another important application area is digital design.

Ex A binary (serial) adder.

As this example indicates, the automaton serves as a bridge b/w the very high-level, functional description of a circuit & its logical implementation

through transistors, gates & flipflops.

* Design of digital circuits.
* Compiler construction
* String Matching
* String processing
* s/w design
* AI, knowledge engg, game theory, CG, Linguistics etc.

Basic notations and terminologies used:

Alphabet: A language consists of various symbols from which the words, statements, etc can be obtained.

* A Language is a subset of the set of all possible strings formed from a given set of symbols.
* There must be a membership criterion for determining whether a particular string in the set.

Grammars:

* A grammar is a formal s/m for accepting or rejecting strings.
* A grammar may be used as the membership criterion for a Language.

Automata:

* An automaton is a simplified, formalized, model of a computer.
* An automaton may be used to compute the membership for for a Language.
* Automata can also compute other kinds of things.

Languages:

Def^n: An alphabet is a finite, nonempty set of symbols. We use Σ to denote this alphabet.

* A string is a finite sequence of symbols from Σ.
* The length of a string s, is denoted by $|s|$, is the no of symbols in it.
* The empty string is the string of length zero.

we usually write it like this: ε (pronounced as epsilon).

* Σ^* denotes the set of all sequences of strings that are composed of zero or more symbols of Σ
  ↳ also called as Kleene star.

  ex: Let $\Sigma = \{0, 1\}$
  $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$
  which is the set of strings of 0's & 1's of any length.

* Kleene Plus: Σ^+
  ↳ denotes the set of all sequences of strings composed of one or more symbols of Σ.

  i.e $\Sigma^+ = \Sigma^* - \varepsilon$

  or $\Sigma^* = \Sigma^+ \cup \varepsilon$

  ex: $\Sigma = \{0, 1\}$
  $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$.

* A language is a subset of Σ^*.

* The concatenation of two strings is formed by forming the sequence of symbols in the first string with the sequence of symbols in the second string.
  If a string S can be formed by concatenating two strings A & B, S = AB, then A is called a prefix of S & B is called a suffix of S.

The reverse of a string S, S^R, is obtained by reversing the sequence of symbols in the string.
For example, if S = abcd, then S^R = dcba.

An automaton is a simple
There is no formal def^{n} for automaton, - instead, ...
various kinds of automata, each with it's own formal definition.

Generally, an automaton
→ has some form of input.
→ has some form of output.
→ has internal states.
→ May or may not have some form of storage.
→ Is hard wired rather than programmable.

* An automaton that computes a boolean function (yes - no) is called an acceptor. Acceptor may be used as the membership criterion of a language.
* An automaton that produces more general o/p (typically a string) is called a transducer.

(Consider an example of abstract data type).
Do you have anything called as stack ADT?
Conceptually we write it, but we dont use it, when actually we implement the stack program which means conceptually it has its existence but physically we dont have anything called as ADT.

In the same way, we have something called as automaton which is an abstract machine.
This machine is conceptually present but physically we do not have this machine.

II class: Language, string, alphabet, ADT, abstract m/c, automata, ε.

def^n, components of automata: I/p, CU, o/p → accept, reject

FA — DFA, NFA, ε-NFA, symbols used in FA, def^n of DFA

Example.

---

* A language consists of various symbols from which the words, statements, etc can be obtained. These symbols are called alphabets. Formally, an alphabet is defined as a finite non-empty set of symbols. The symbol Σ denotes the set of alphabets of a language.

Ex: 1. Σ = {0, 1} denoted the set of alphabets of m/c language.

Ex 2: Σ = { a, b, ... z, A, B, ... Z, 0, 1, ... 9, #, (,), {, }, <, >, !, [,], ... } represent the alphabets of C language.

String:

The sequence of symbols obtained from the alphabet of a language is called a string. Formally, a string is defined as a finite sequence of symbols from the alphabet Σ.

Let Σ = {0, 1} is set of alphabets.

The various strings that can be obtained from Σ are {0, 1, 00, 01, 10, 11, 0101, 1010, ...}

An empty string is denoted by the symbol ε (pronounced as epsilon) & note that ε ∉ Σ i.e ε is not part of Σ.

Normal Notations:

* The symbol ε is used to denote an empty string.
* The lowercase letters a, b, c, etc along with the symbols such as +, -, (,), & & so on are used to denote the symbols in Σ.
* The lowercase letters such as u, v, w, x, y, z are normally used to indicate the strings. For example, we can write w = 010101
where the symbols 0 & 1 are in Σ & the letter w denotes the string with a specific value.

Formal definition of a language:
A language can be defined as a set of strings obtained from Σ^* where Σ is set of alphabets of particular language. Formally, a language L over Σ is subset of Σ^* which is denoted by $L \subseteq \Sigma^*$.

Finite Automata: Finite automata (FA) are computing devices that accept / recognize regular languages. The FA acts as mathematical models & are used to study the abstract machines or abstract computing devices. The various operations of FA can be simulated by simple computer programs. The FA on reading the input string, may accept the string or reject the string. Using this abstract model, the behavior of the actual s/m can be understood & build to perform various activities. Any FA has 3 components as shown below:

1. Input file
2. Control Unit
3. Output.

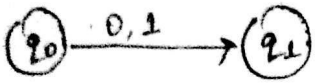
* Input tape: The I/p tape is divided into each of which can hold one symbol. The string to be processed is stored in these cells.

* Control Unit: This unit may be in any of the states denoted by q_0, q_1, q_2, & ...so on. The state q_0 is considered as the start state. The machine has atleast one final state. Based on the current I/p symbol & the current state of the machine, the state of the machine can change.

* Output: O/p may be accept or reject. When end of I/p is encountered, the control unit may be in accept or reject state.

Symbols used in FA:

	Symbol	Meaning
1)	(q_0)	A circle is used to represent a state. Here, q_0 is a state of the machine.
2)	$\rightarrow (q_0)$	A circle with an arrow which is not originating from any node represents the start state of machine.
3)	$((q_0))$	Two circles are used to represent a final state.
4)	$(q_0) \xrightarrow{1} (q_1)$	There is a transition from state q_0 on I/p symbol 1 to state q_1. This can be written as: $\delta(q_0, 1) = q_1$.
5)	$0 \circlearrowleft (q_0)$	An arrow with Label 0 starts & ends in q_0. i.e $\delta(q_0, 0) = q_0$



i.e $\delta(q_0, 0) = q_1$

$\delta(q_0, 1) = q_1$

Different types of finite automata:

1) Deterministic Finite Automata (DFA)

2) Non- " " ——— (NFA)

3) ——— " ——— " ——— with ε moves (ε-NFA)

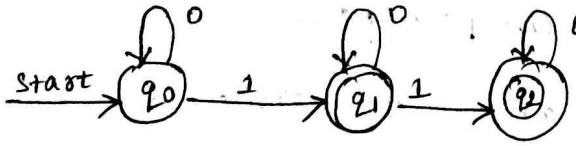
DFA:

Deterministic: There is exactly one transition for every i/p symbol from the state. So, it is possible to determine exactly to which state the m/c enters into after consuming the i/p symbol. So, the m/c is deterministic.

Finite: Has finite no of states & arcs. So, it is deterministic & finite.

Automaton: Automaton is a m/c which may accept the string or reject the string. So, it is deterministic finite automaton.

Ex:



final state.

From the above figure, observe the following components of DFA.

① States: The above DFA has 3 states. i.e $Q = \{q_0, q_1, q_2\}$

② Input alphabets: $\Sigma = \{0, 1\}$

③ Transitions: $\delta(q_i, a) = q_j$

i.e $\delta(q_0, 0) = q_0$

$\delta(q_0, 1) = q_1$

$\delta(q_1, 0) = q_1$

$\delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_2$

$\delta(q_2, 1) = q_2$ i.e $\delta(Q \times \Sigma)$ to Q

which is the cross product of $Q = \{q_0, q_1, q_2\}$ &

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

Here δ is a transition function.

ex: $\delta(q, a) = p$

where

δ: is a fn called transition fn.

q: is the first parameter representing the current state of the m/c

a: is the second parameter representing the current i/p symbol read.

p: is the next state of m/c which is returned by the transition fn.

* Start state:
* Final state:

∴ From the above discussion, it is obvious that DFA has 5 components.

(states, i/p alphabets, transitions, start state, final state)

Q Σ δ q_0 F

∴ DFA $M = (Q, \Sigma, \delta, q_0, F)$

DFA is 5 tuple or quintuple indicating 5 components

where M is the name of the machine. It can be called by any name.

Q is non-empty, finite set of states

Σ is ——— ; finite set of i/p alphabets.

δ is a transition fn which is mapping from $Q \times \Sigma$ to Q i.e $\delta : Q \times \Sigma$ to Q

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accepting or final states.

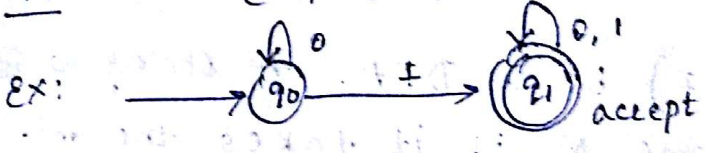
Simple Notations for DFA's:

A DFA can be specified using simpler & more effective notations

1. Transition diagram (Transition graph)
2. Transition table.

TD: $M = (Q, \Sigma, \delta, q_0, F)$

Ex:



② Transition table:

The ←——→ for DFA is defined as a conventional tabular representation of a transition fn such as δ which takes two arguements & returns a value.

with:

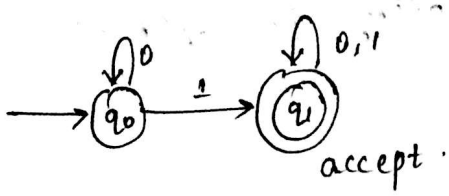
1. The rows of the table correspond to the states of DFA ~~define~~ obtained from Q.
2. The columns correspond to the I/P symbols obtained from Σ.

Note: There is one row for each state, & one column for each I/P.

3. If q is the current state of DFA, & a is the current I/P symbol, the value returned from $\delta(q, a)$ represent the next state of DFA & is entried in row q & column a.

* The start state is marked with an arrow.
* The final state is marked with a star.

ex:



Transition Diagram

δ	0	1
$\rightarrow q_0$	q_0	q_1
$*q_2$	q_1	q_1

Transition table.

Language accepted by a DFA:

The language accepted by DFA is formally defined as shown below:

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. A string w is accepted by the machine M, if it takes the m/c from initial state q_0 to final state. i.e $\delta^*(q_0, w)$ is in F. Thus, the language accepted by DFA represented as $L(M)$ can be formally written as:

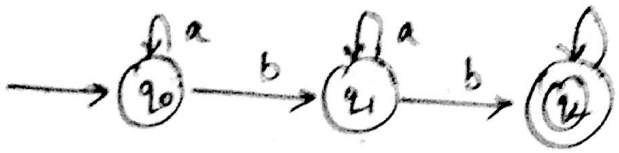
$$L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta^*(q_0, w) \text{ is in } F\}.$$

what language is rejected by DFA ?

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The non-acceptance of a language indicates that after the string w is processed, the DFA will not be in the final state.

$$\text{i.e } L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta^*(q_0, w) \text{ is not in } F\}.$$

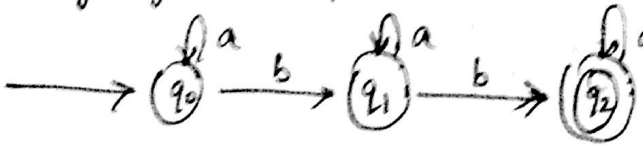
ex:



accept.

string: abaa → rejected by above DFA.

Language accepted by DFA example:



accept

string: abab is accepted by above DFA.

Extended transition function of DFA to strings:

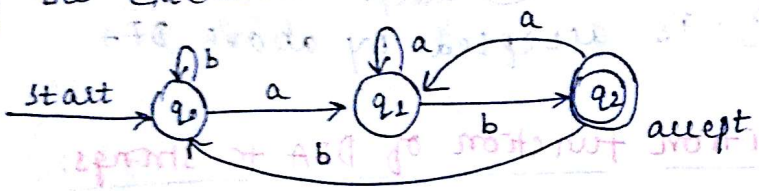
Note: The transition $\delta(q, a) = p$ accepts 2 parameters namely state q & i/p symbol a as the parameters and returns a state p which is the next state of the machine. But, if there is a change of state from state q to state p on i/p string w, then we use the extended transition fn denoted by δ^*.

Sy: The extended transition fn. δ^* describes what happens to a state of machine when the i/p is a string (sequence of symbols)

- Kleene star
- Kleene Plus

BASIS: $\delta^*(q, \varepsilon) = q$. this indicates that If the m/c is in state q & read no i/p, then the machine is still in state q.

Ex: Let us see, what are the moves made by the following DFA while processing the string "abaab" using the extended transition fn.



Sol^n: The moves made by the DFA for the i/p string abaab using δ^* is obtained from ε & taking prefix of abaab in increasing size as shown below:

① For prefix ε: $\delta^*(q_0, \varepsilon) = q_0 \longrightarrow ①$

② For prefix a: $\delta^*(q_0, a) = \delta(\delta^*(q_0, \varepsilon), a)$ sub ①

$= \delta(q_0, a)$

$= q_1 \longrightarrow ②$

③ For prefix ab:

$\delta^*(q_0, ab) = \delta(\delta^*(q_0, a), b)$ sub ②

$= \delta(q_1, b)$

$= q_2 \longrightarrow ③$

④ For prefix aba:

$\delta^*(q_0, aba) = \delta(\delta^*(q_0, ab), a)$ sub ③

$= \delta(q_2, a)$

$= q_1 \cdots ④$

⑤ For prefix abaa:

$\delta^*(q_0, abaa) = \delta(\delta^*(q_0, aba), a)$ sub ④

$= \delta(q_1, a)$

$= q_1 \cdots ⑤$

* For prefix abaab: $\delta^*(q_0, abaab)$

$= \delta(\delta^*(q_0, abaa), b)$ sub (5)

$= \delta(q_1, b)$

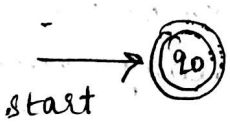
$= q_2$.

After the string abaab, the m/c is in state q_2 which is the final state.

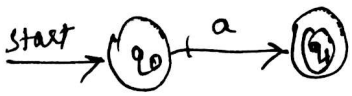
ex 1: DFA to accept empty language $L = \{\phi\}$.

X. 

2. DFA to accept an empty string $L = \{\varepsilon\}$

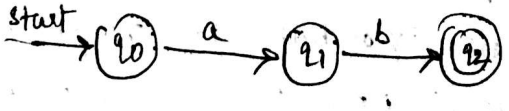


3. DFA to accept exactly one a



i.e To accept one symbol a we require 2 states.

4. DFA to accept string ab



To accept 2 symbols such as ab, we require 3 states. In general, to accept n symbols, we require n+1 states.

The various steps to be followed are:

Step1: Identify Σ and the minimum string & write the DFA to accept the minimum ~~DFA~~ string. This is called initial DFA

Step2: Identify other transitions not defined in step1. For example, If $\delta(q_0, a)$ is known in step1 & $\delta(q_0, b)$, $\delta(q_1, a)$, $\delta(q_2, b)$ are not known, then we write,

$\delta(q_0, b) = ?$

$\delta(q_1, a) = ?$

$\delta(q_2, b) = ?$

Step3: construct the DFA using transitions in step1 & step 2. show, the formal DFA along with transition diagram.

Worked Examples:

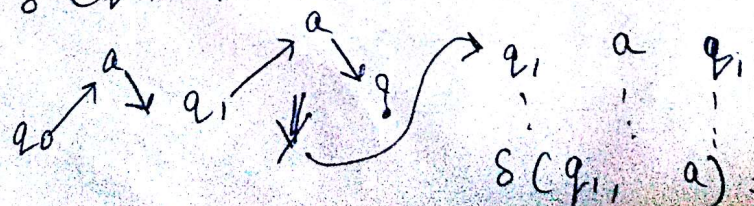
① Draw a DFA to accept strings of a's & having atleast one a.

Step1: $\Sigma = \{a\}$, Min. string = "a" with one symbol. since it has to accept one symbol, it requires 2 states q_0 & q_1. & the initial DFA is shown below:

$\Sigma = \{a\}$

Step2: Identify the transitions not defined in step1.

i) $\delta(q_1, a) = ?$



$\delta(q_1, a) = q_1$.

Step 3: ∴ The DFA $M = \{Q, \Sigma, \delta, q_0, F\}$

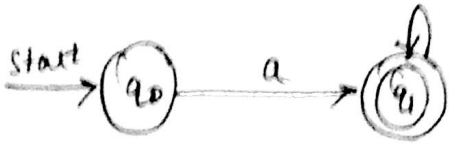
$Q = \{q_0, q_1\}$

$\Sigma = \{a\}$

q_0 is the start state

$F = \{q_1\}$

δ is shown below using the TD & TT.



δ	a
→ q_0	q_1
* q_1	q_1

TD

TT

Note: The language accepted by above DFA can also be written as: $L = \{a, aa, aaa, aaaa, \ldots\}$

or

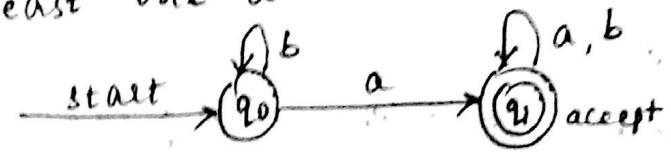
$L = \{a^n \mid n \geq 1\}$ or $L = \{w : n_a \geq 1, w \in \{a\}^*\}$

---

(2) Draw a DFA to accept strings of a's & b's having atleast one a.

Min string is a.

$\Sigma = \{a\}$

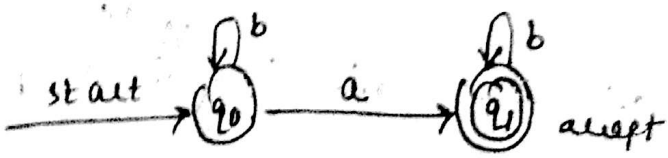


$L = \{w : n_a(w) \geq 1, w \in \{a, b\}^*\}$

(3) DFA which accepts strings of a's & b's having exactly one a.

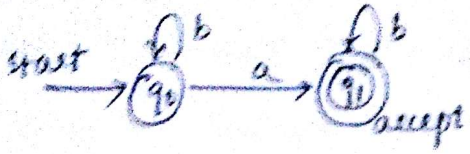
Here $\Sigma = \{a, b\}$ & min string = 'a'.

ie

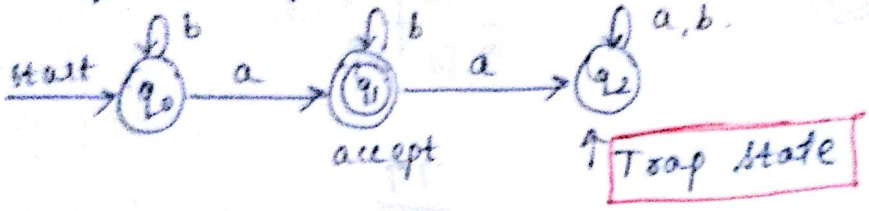


$L = \{w : n_a(w) = 1, w \in \{a, b\}^*\}$

Consider the DFA



In the above DFA, there is no transition from state q_1 for the I/P symbol a. But, we can include a transition from state q_1 for the I/P symbol a to a non-final reject state as shown below:



state q_2 is called a trap state.

* What is a trap state?

A state for which there exists transitions to itself for all the I/P symbols chosen from Σ is called a trap state. It is also called dead state.

* There is no escape from this state. The m/c is trapped in this state & hence the name trap state.

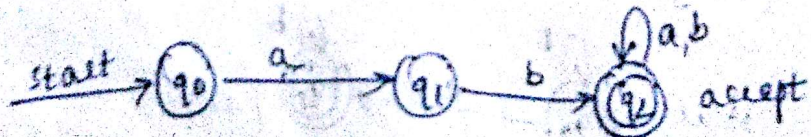
A trap state can be either non-final or final state

Non final - Dead state.

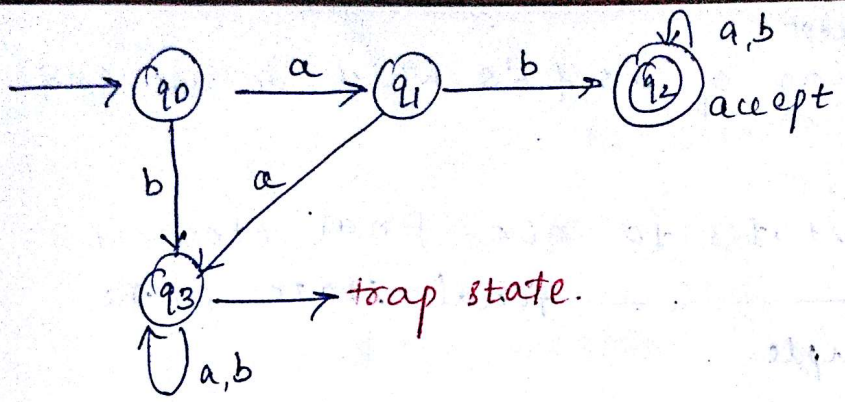
Final - not a dead state since the string is accepted at this state.

(4) Obtain a DFA to accept strings of a's & b's starting with the string ab.

Here $\Sigma = \{a, b\}$ & min string: "ab".

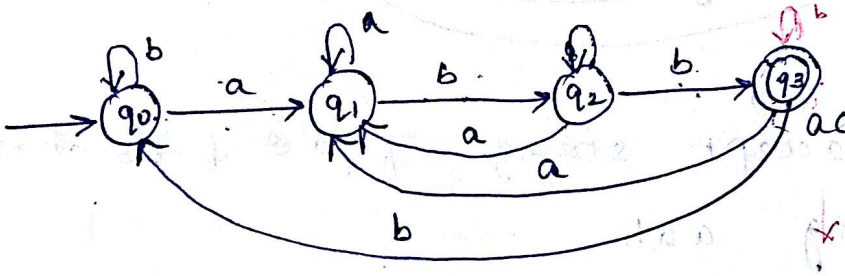


Q.1



⑤ DFA to accept string of a's & b's ending with the string abb.

$\Sigma = \{a, b\}$ min string = abb.



baab.
baaabab.
babbabb
babbabbb
babbbabb

$\therefore L = \{ (a+b)^n abb : n \geq 0 \}$.

Assignment:

① The set of all strings beginning with 101

② " " " containing 1101 as a substring.

③ " " with exactly 3 consecutive 0's.

④ " " such that the no of 1's is even & the no of 0's is a multiple of 3.

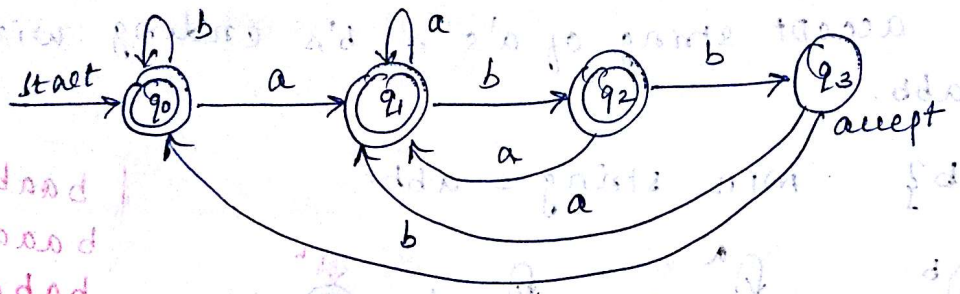
⑤ The set of all strings not containing 110

⑥ " " that begin with 01 & end with 11

⑦ The set of all strings which when interpreted as a binary integer is a multiple of 3.

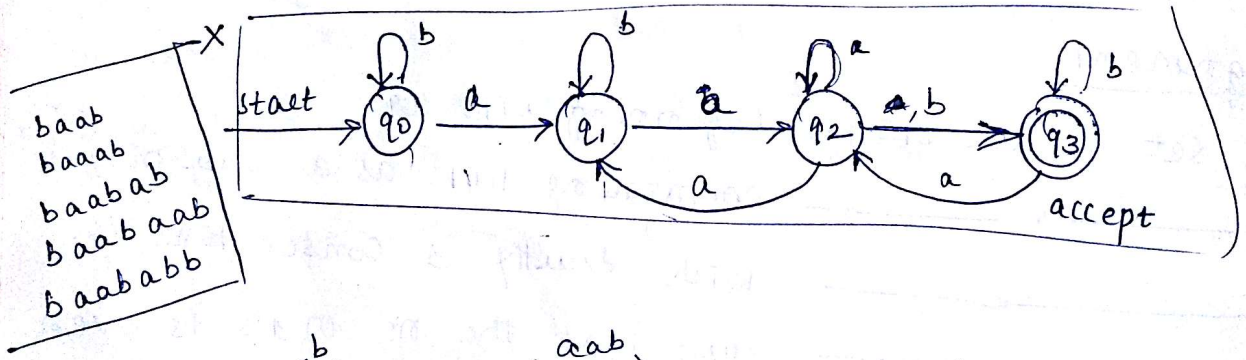
⑥ Draw a DFA to accept string of a's & b's which do not end with string abb.

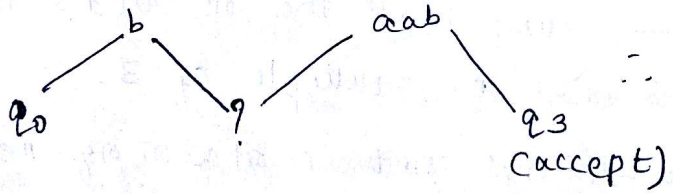
Soln: Make final states to non-final states & non-final states to final states in the previous example.



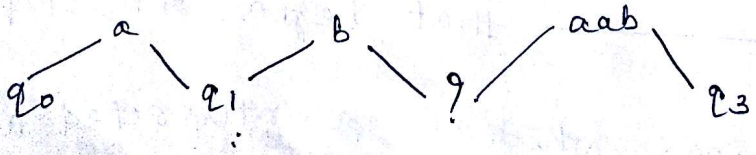
⑦ DFA to accept strings of a's & b's having a substring aab.

Soln: $\Sigma = \{a, b\}$ Min. string = aab.



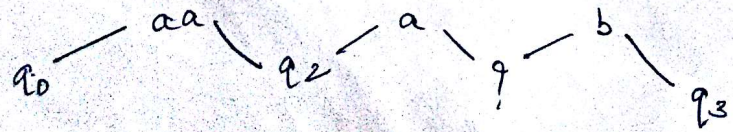
① 

$\therefore \delta(q_0, b) = q_0$

② 

ab should not be accepted but accept abaab

$\therefore \delta(q_1, b) = q_1$

③ 

$\therefore \delta(q_2, a) = q_2$

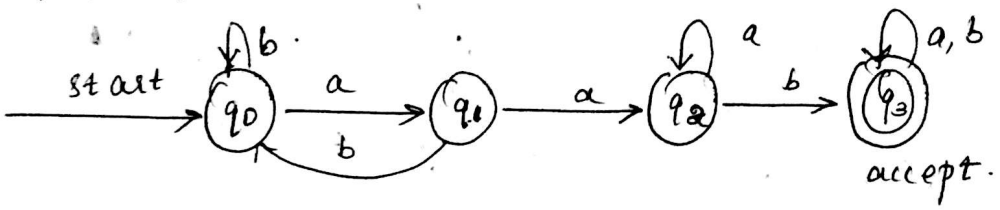
④ $q_0 \xrightarrow{aab} q_3 \xrightarrow{a} q_3$

ex. aaba should be accepted

$\therefore \delta(q_3, a) = q_3$.

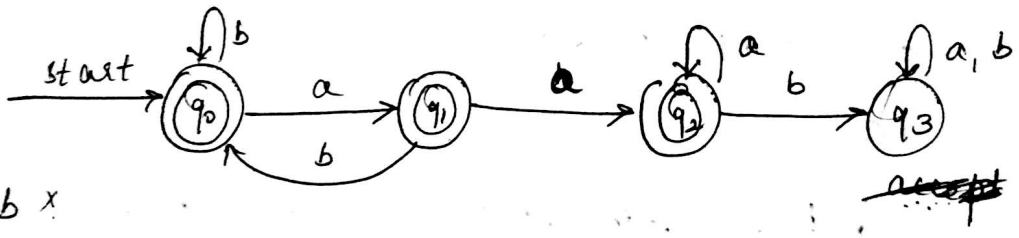
⑤ aabb should be accepted. $\therefore \delta(q_3, b) = q_3$

Final DFA is



⑥ DFA to accept string of a's & b's except those having the string aab.

Sol^n: Final ⟶ Non-final & vice versa.



babaab ✗

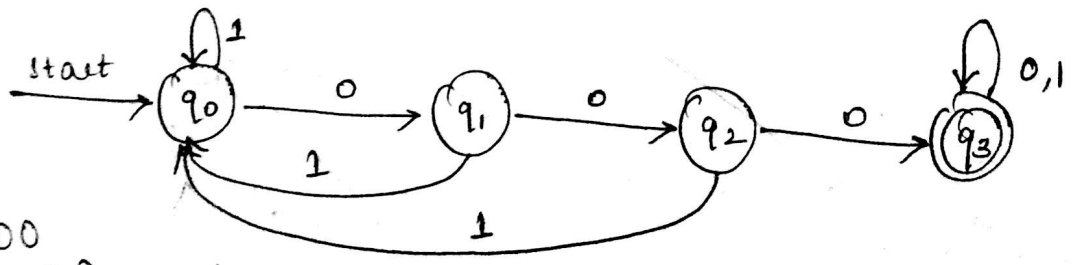
baba ✓

babaababa ✗

ababba ✓

⑦ DFA to accept string of 0's & 1's having three consecutive 0's.

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



1000

101000

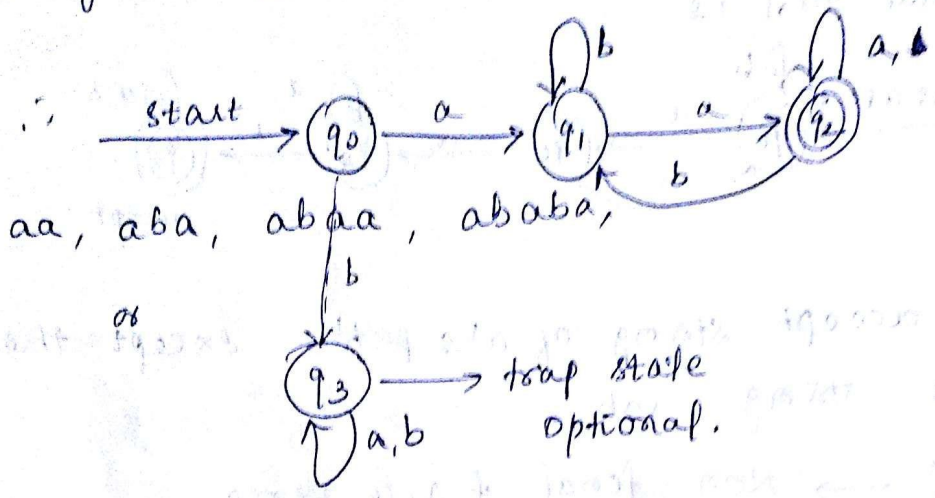
1001000

101001000

10001 & 10000

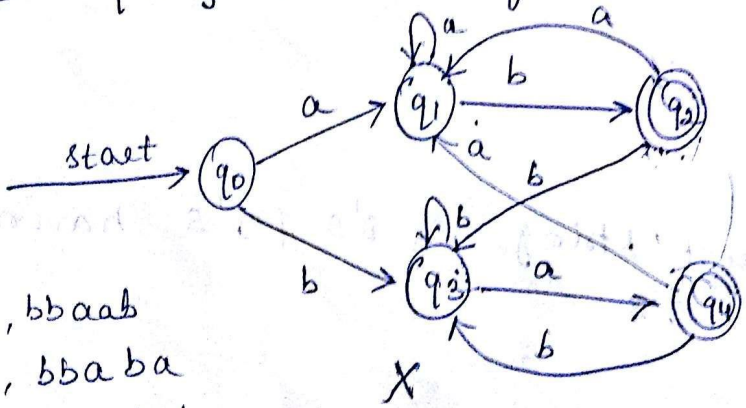
(10) DFA to accept strings of a's & b's such that $L = \{awa \mid w \in (a+b)^n \text{ where } n \geq 0\}$

Sol^n: DFA which accepts strings of a's & b's starting with one a & ending with one a with minimum string is aa

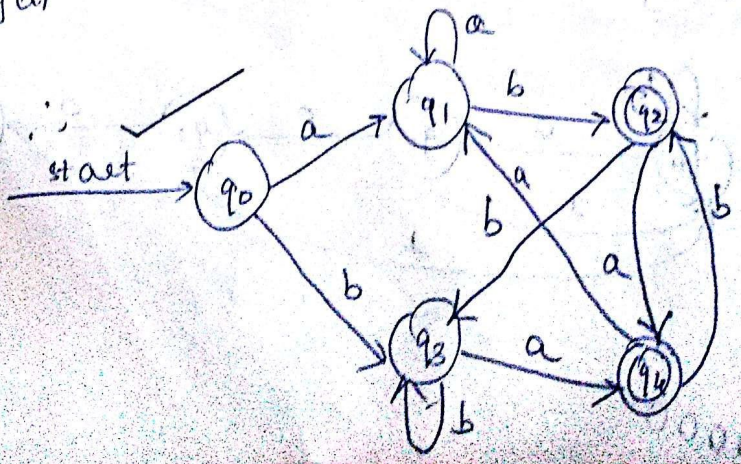


(11) DFA to accept strings of a's & b's ending with ab or ba.

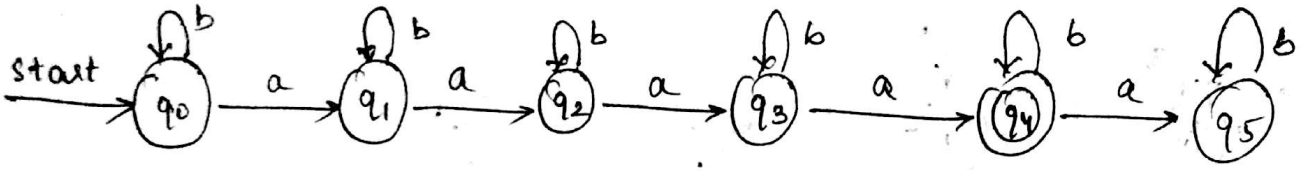
$\therefore \Sigma = \{a, b\}$. & Min. string is ab or ba.



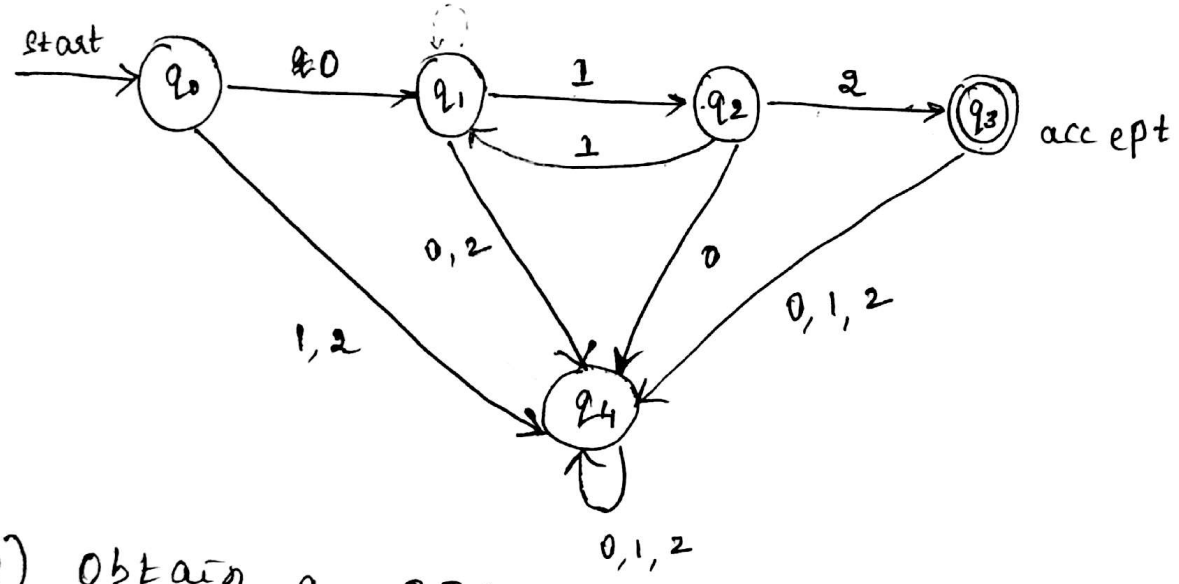
aabba, bbaab
aabab, bbaba
aba, bab ✗ ject



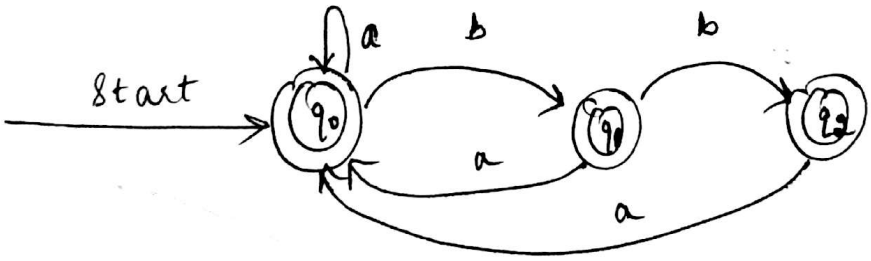
12) Obtain a DFA to accept strings of a's & b's having four a's.



13) Obtain a DFA to accept strings of 0's, 1's, & 2's beginning with a 0 followed by odd no of 1's & ending with a 2.



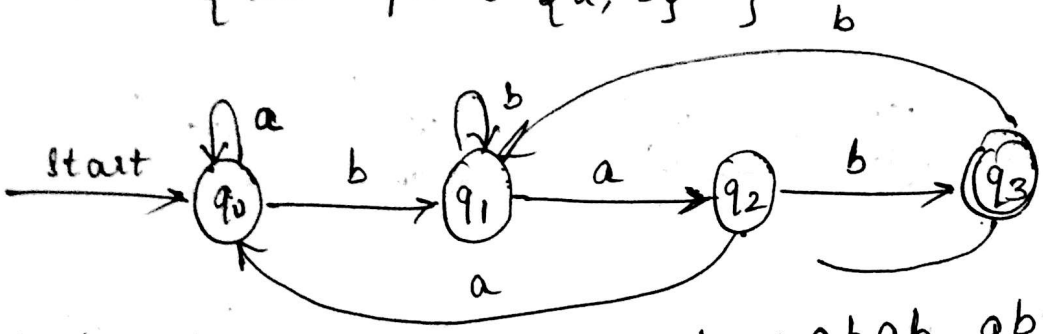
14) Obtain a DFA to accept strings of a's & b's with atmost two consecutive b's.



15) Obtain a DFA to accept strings of 0's & 1's starting with atleast two 0's & ending with atleast two 1's.

16) Obtain a DFA to accept the language

$L = \{ wbab \mid w \in \{a, b\}^* \}$



abab, abbab, abbbab, abaaabab, ababab,

abababbab

—•—•—•—

NFA (Non-Deterministic Finite Automata):

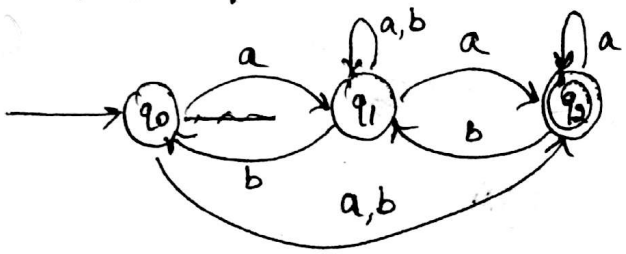
Nondeterminism means a choice of moves for an automaton. Rather than prescribing a unique move in each situation, we allow a set of possible moves. Formally, we achieve this by defining the transition function so that its range is a set of possible states.

* Computers are completely deterministic machines.
* The state of the computer cannot be predicted from the input and initial state. We cannot find a computer which is non-deterministic. In such case, the question is why ~~###~~ NFA?

The various advantages of NFA are as follows:

* Constructing most of the NFA's is very easy.
* A "non-deterministic" finite automaton has the ability to guess something about its i/p.
* It is more powerful than DFA.
* It has the power to be in several states at once.
* An NFA is an efficient mechanism to describe some complicated languages concisely.

Consider the following example:



* **States:** The NFA has 3 states q_0, q_1, & q_2 & can be represented as: $Q = \{q_0, q_1, q_2\}$

Note: The power set of Q is denoted by 2^Q which is set of subset of set Q. This is denoted as shown below:

$2^Q = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$

+ I/p alphabets: $\Sigma = \{a, b\}$

+ Transition Diagram:

Transition is nothing but change of state from current state after consuming an i/p symbol.

If there is a change of stage from q_i to q_j & q_i to q_k, then we write,

$\delta(q_i, a) = \{q_j, q_k\}$.

The transition diagram of above NFA is shown below:

current state	Input	Next state	Representation
q_0	a	q_1, q_2	$\delta(q_0, a) = \{q_1, q_2\}$
q_0	b	q_2	$\delta(q_0, b) = \{q_2\}$
q_1	a	q_1, q_2	$\delta(q_1, a) = \{q_1, q_2\}$
q_1	b	q_0, q_1	$\delta(q_1, b) = \{q_0, q_1\}$
q_2	a	q_2	$\delta(q_2, a) = \{q_2\}$
q_2	b	q_1	$\delta(q_2, b) = \{q_1\}$

δ Q X Σ to 2^Q (power set)

* Start state : 

* Final state : 

* A NFA is ~~short~~ ~~an~~ a 5 tuple or quintuple:

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

Where M is the name of the m/c. It can also be called by any name.

Q is non-empty, finite set of states

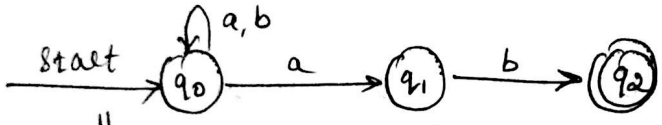
Σ is " " " i/p alphabets.

δ: $Q \times \Sigma$ to 2^Q. Based on the current state & i/p symbol, the m/c enters into one or more states.

$q_0 \in Q$ is the start state.

$F \subseteq Q$ is set of final states.

ex: NFA to accept strings of a's & b's ending with ab:



Transition table

δ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
* q_2	ϕ	ϕ

⇒ In the TT of NFA, each entry in the table should be denoted by a set. Also, when there is no transition from a given state on i/p symbol, make an entry ϕ indicating an empty set.

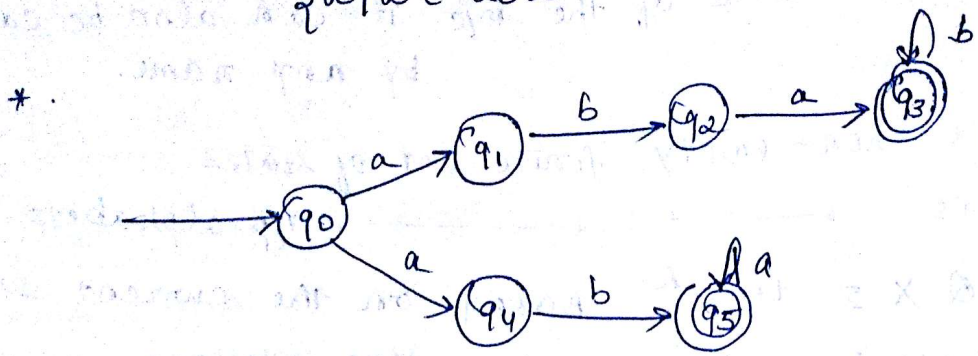
Language accepted by a NFA:

$L(M) = \{w \mid w \in \Sigma^* \text{ and } \delta^*(q_0, w) \text{ is in } F\}$

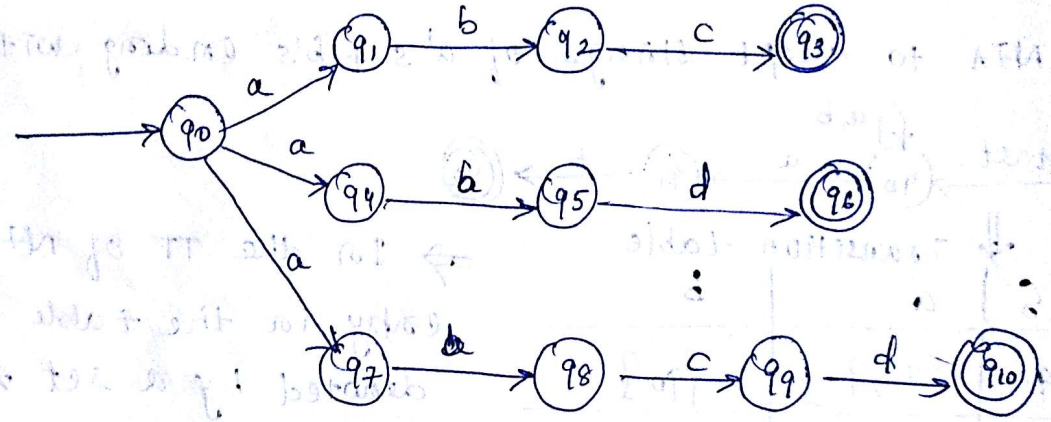
~~ex~~:

① Obtain an NFA to accept the language

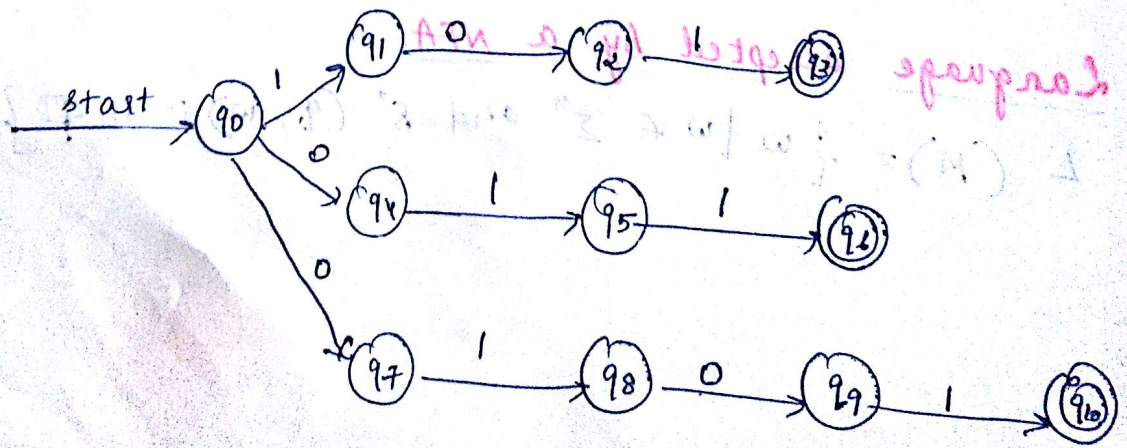
$L = \{w \mid w \in aba^n \text{ or } aba^n \text{ where } n \geq 0\}$



② NFA to recognize the following set of strings abc, abd & aacd.



③ NFA to recognize the following set of strings 0101, 101 & 011.



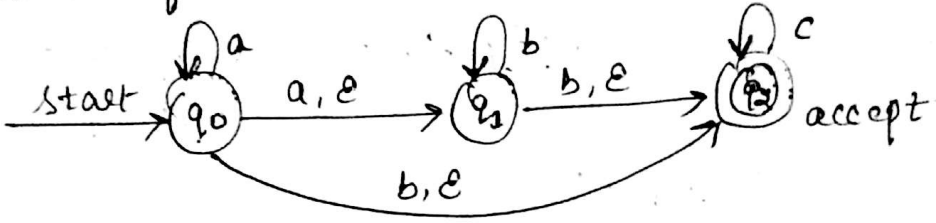
ε - NFA

Finite Automata with Epsilon (ε) Transitions:

An NFA with zero or more ε-transitions is called an ε-NFA.

Defn: A transition with an empty i/p string is called an ε-transition. That is, if there is a transition from one state to another state without any i/p (i.e no i/p implies empty string denoted by ε) is called ε-transition.

Consider the finite automaton shown below:



From the above FA, observe the following points:

① In the above FA, we have more than two transitions from state q_0 with i/p symbol a. So, it is an NFA.

② There is a transition from state q_0 to q_1 with ε as the i/p. There is another transition from state q_1 to q_2 with ε as the i/p. So, the above FA is an NFA with ε-transitions, & hence it is an ε-NFA.

Note: If zero or more ε-transitions are present in a FA, then it is an ε-NFA.

For the above NFA, TT is as shown below:

Current state	I/p	Next state	Representation
q_0	a	$\{q_0, q_1\}$	$\delta(q_0, a) = \{q_0, q_1\}$
q_0	b	$\{q_2\}$	$\delta(q_0, b) = \{q_2\}$
q_0	c	ϕ	$\delta(q_0, c) = \phi$

q_0 ε $\{q_1, q_2\}$ $\delta(q_0, a) = \{q_1, q_2\}$

q_2 ε ϕ $\delta(q_2, \varepsilon) = \phi$

$\therefore \delta$ is $(Q \times (\Sigma \cup \varepsilon))$ to 2^Q

* The ε-NFA is 5-tuple or quintuple indicating 5 components:

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

where * Q is non-empty, finite set of states.

* Σ is ——————— i/p alphabets.

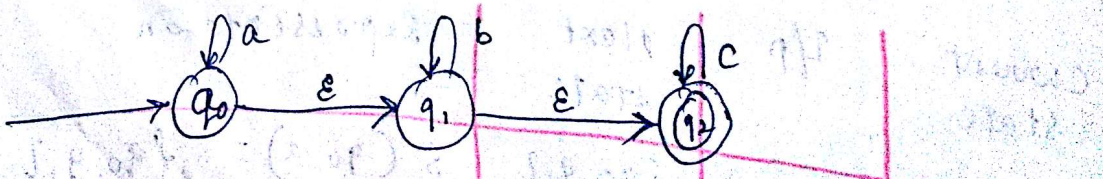
* δ : Q × (Σ ∪ ε) to 2^Q. i.e δ is a transition function which is a mapping from Q × (Σ ∪ ε) to 2^Q. Based on the current state there can be a transition to other states with or without any i/p symbols.

* $q_0 \in Q$ - is the start state.

* $F \subseteq Q$ - is set of accepting or final states.

* Extended transition fn

Ex①. obtain an NFA which accepts strings consisting of zero or more a's followed by zero or more b's followed by zero or more c's.



* Conversion from ε-NFA to DFA:

Let $M_E = (Q_E, \Sigma, \delta_E, q_0, F_E)$ be an ε-NFA where Q_E is set of finite states, Σ is set of i/p alphabets, δ_E is transition from $Q_E \times \{\Sigma \cup \varepsilon\}$ to 2^{Q_E}, q_{0E} is the start state, & F_E is the set of final states.

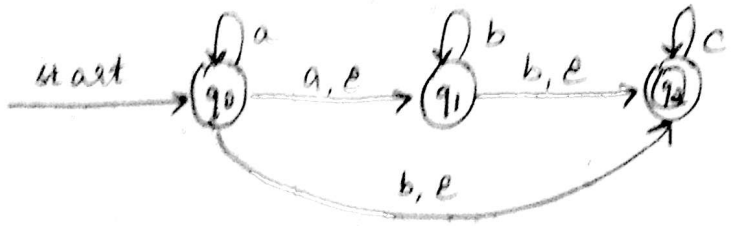
The equivalent DFA is $M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$ can be obtained as shown below with the help of an example.

* Def^n of ε-CLOSURE:

The ε-CLOSURE of q denoted by ECLOSE (q) is the set of all states which are reachable from q on ε-transitions only. It is recursively defined using recursion as shown below.

* State q is in ECLOSE (q) i.e ECLOSE (q) = q.
* If ECLOSE (q) contains p and if there is a transition from state p to state r labelled ε, then state r is also in ECLOSE (q).

Ex:



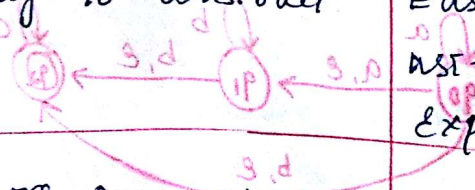
ECLOSE $(q_0) = \{q_0, q_1, q_2\}$

ECLOSE $(q_1) = \{q_1, q_2\}$

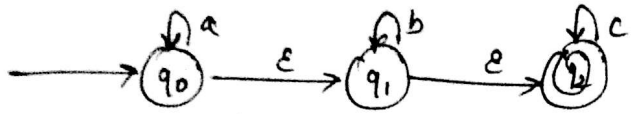
ECLOSE $(q_2) = \{q_2\}$

* Difference b/w DFA, NFA & ε-NFA:

DFA	NFA	ε-NFA
1) Write Defⁿ of DFA $M = (Q, \Sigma, \delta, q_0, F)$ where $\delta: Q \times \Sigma$ to Q	NFA Defⁿ $\delta: Q \times \Sigma$ to 2^Q	ε-NFA Defⁿ $\delta: Q \times (\Sigma \cup \varepsilon)$ to 2^Q
2) There can be zero or one transition from a state on an i/p symbol.	There can be zero, one, or more transitions from a state on an i/p symbol.	There can be zero, one or more transitions from a state with or without giving any input.
3) More no of transitions	Less no of transitions.	Relatively more transitions when compared to NFA.
4) Difficult to construct	Easy to construct	Easy to construct using regular expressions.
5) Less powerful since at any point of time it will be in only one state.	More powerful than DFA since at any point of time it will be in more than one state.	More powerful than NFA since at any point of time it will be in more than one state with or without giving any input

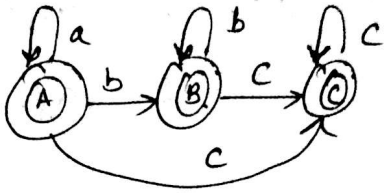


ex 2)



Soln: TT:

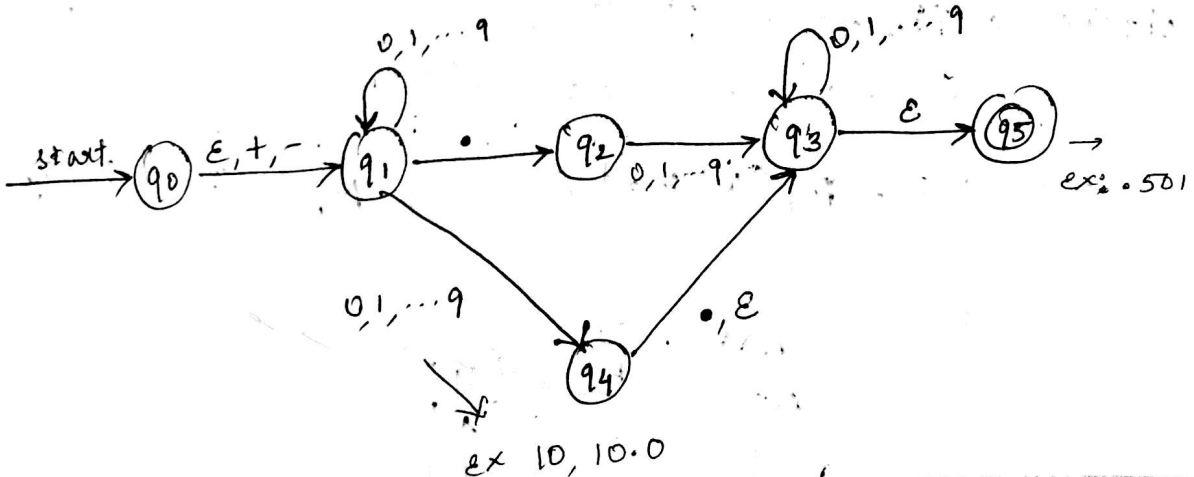
	δ	a	b	c
→	*A	A	B	C
	*B	ϕ	B	C
	*C	ϕ	ϕ	C



ex. 3) Obtain an NFA with ε-transitions to accept decimal no's {("Obtain δ* (q0, 4.7)"}.

↓ consisting of

1) an optional + or - sign
2) a string of digits
3) a decimal point .
4) another string of digits. Either string of digits, or the string (2) can be empty, but atleast one of the 2 strings of digits must be nonempty.



TT

for the above ex. 2.

A → {q0, q1, q2}

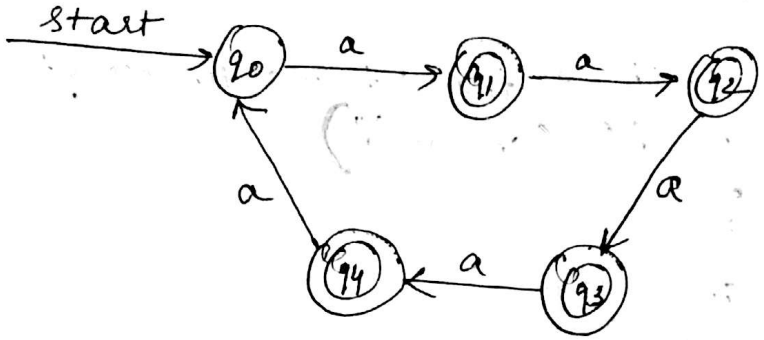
B → {q1, q2}

C → {q2}

4) ~~E NFA to DFA~~

DFA problems

① DFA to accept language $L = \{w : |w| \bmod 5 \neq 0\}$ over $\Sigma = \{a\}$.

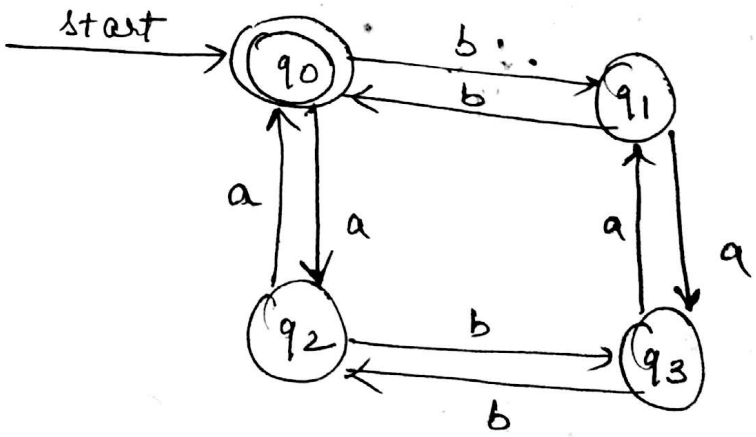


② DFA to accept language $L = \{w : |w| \bmod 5 = 0\}$

Sol^n: In the above DFA, just interchange final & non-final states.

3) obtain a DFA to accept strings of a's & b's having even no of a's & even no of b's.

i.e $L = \{N_a(w) \bmod 2 = 0 \text{ \& } N_b(w) \bmod 2 =$



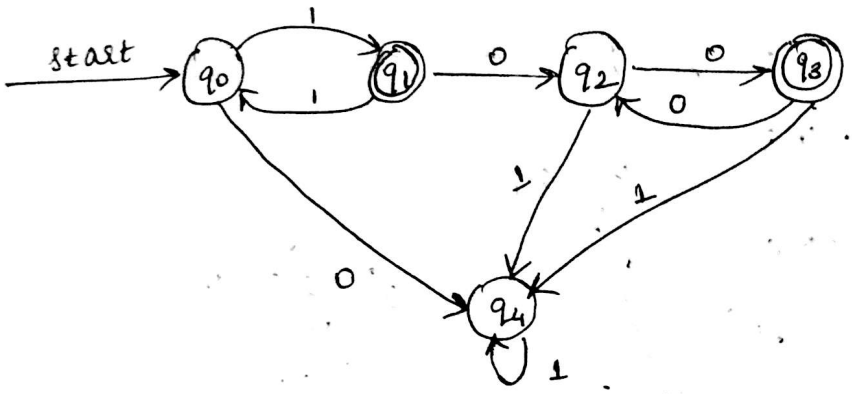
4) odd no of a's & even no of b's → q_2 final sta

even no of a's & odd no of b's → q_1 ——

odd no of a's & odd no b's → q_3 ——

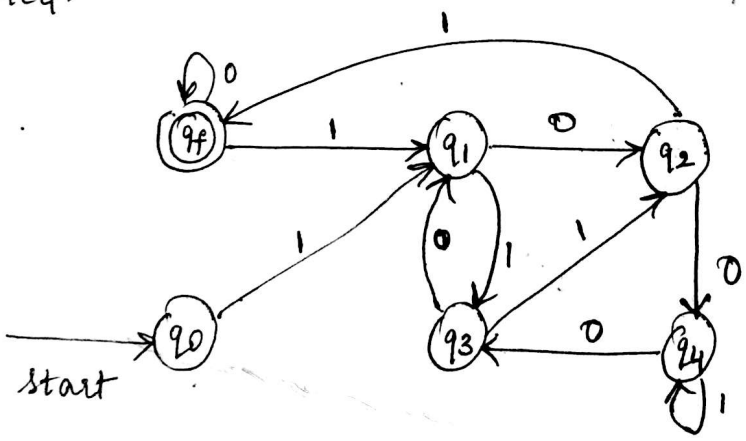
Extra problems on DFA:

(1) Draw a DFA to accept the language.
L : {w: w has odd no of 1's and followed by even no of 0's}.



Divisible by K-problems:

* Obtain a DFA which accepts the set of all strings beginning with a 1 that when interpreted as a binary integer, is a multiple of 5.
For ex: 101, 1010, 1111 etc are multiples of 5. Note that 0101 is not beginning with 1 & it should not be accepted.



101
1010
1111

* Draw a DFA to accept decimal strings divisible by 3.

q_0 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

possible remainders are:

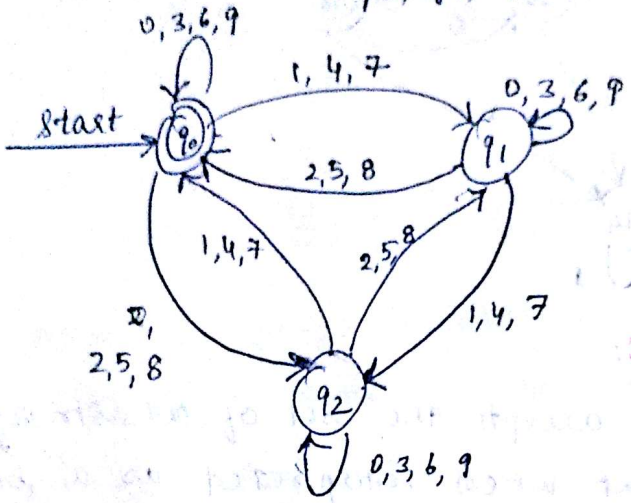
{0, 3, 6, 9} → with 0 as the remainder

{1, 4, 7} → " — 1 "

{2, 5, 8} → " — 2 "

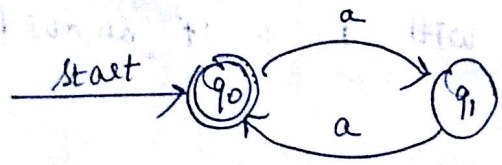
↓ which implies

q_0, q_1, q_2 are the states.

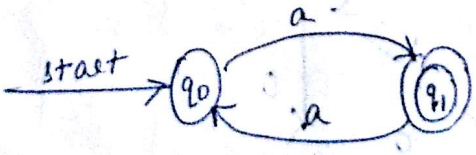


3) 123 (41
121
·03
3
0
122 ✗

* DFA to accept even no of a's.



* Odd no of a's



Equivalence and Minimization of automata:

The language generated by a DFA is unique. But, there can exist many DFA's that accept the same language. In such cases, the DFA's are said to be equivalent. During computations, it is desirable to represent the DFA with fewer states since the space is proportional to the number of states of DFA. For storage efficiency, it is required to reduce the no of states & hence it is required to minimize the DFA. This can be achieved first by finding the distinguishable & indistinguishable states.

Def 1: Two states p & q of a DFA are equivalent (indistinguishable) if and only if $\delta(p,w)$ & $\delta(q,w)$ are final states or both $\delta(p,w)$ & $\delta(q,w)$ are non-final states for all $w \in \Sigma^*$.

i.e $\delta^*(p,w) \in F$ & $\delta(q,w) \in F$
& $\delta(p,w) \notin F$ & $\delta(q,w) \notin F$
} equivalent or indistinguishable

$\delta(p,w) \in F$ & $\delta^*(q,w) \notin F$ or vice versa, then the states p & q are said to be distinguishable or not equivalent.

One method for reducing the states of a DFA is based on finding and combining indistinguishable states.

* Following is a method for finding pairs of distinguishable states and indistinguishable state
(also called as Mark procedure)

* The Table filling algorithm is used to find the states that are distinguishable & indistinguishable.

Example 1:

Obtain the distinguishable table for the automato then minimize the states of following DFA.

Sol^n: The DFA can be minimized using table filli algorithm as shown below.

*. The various states of given DFA are:

A, B, C, D, E, F, G, H

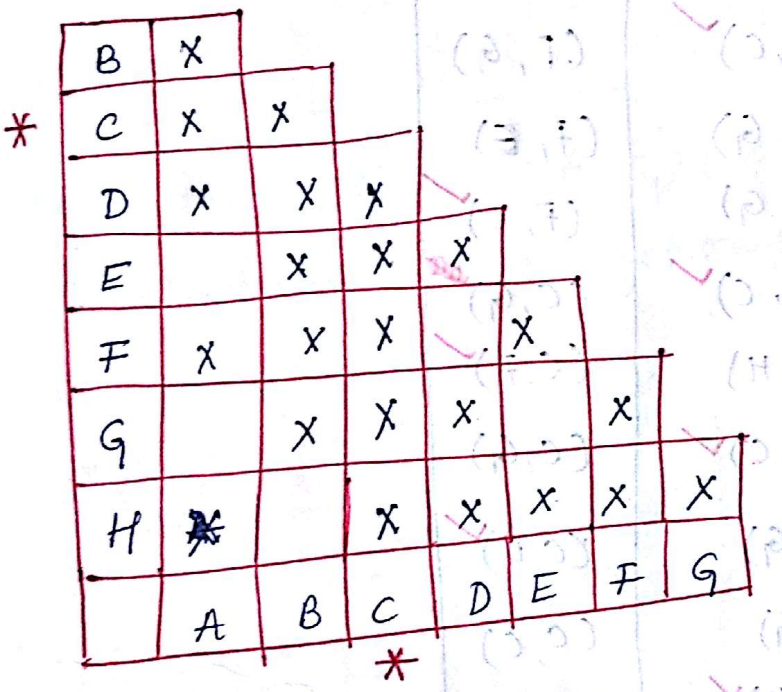
δ	a	b
→ A	B	F
B	G	C
* C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

Step 1: Obtain the various pairs of states.
The various pairs from the above states be obtained by drawing the table shown be

* Vertically, we write all the states from second state to the last state. In this case, we write the states B, C, D, E, F, G & H vertically.

* Horizontally, we write all the states from first state to last but one state. In this case, we write all the states A, B, C, D, E, F, & G.

* Then draw the vertical & horizontal lines as shown below in the table.



⟹ Table-1.

1) Initial Horizontal Markings: Since state C is the final state, the pairs (A,C) & (B,C) has one final state & other non-final state. So we mark the pairs (A,C) & (B,C) horizontally.

2) Initial Vertical Markings: Since state C is the final state, the pairs (H,C), (G,C), (F,C), (E,C) & (D,C) has one final state & non-final state. So they are marked

Step 2: For each pair (p, q) & for each $a \in \Sigma$, find $\delta(p, a) = r$ & $\delta(q, a) = s$. If the pair (r, s) is already marked as distinguishable then the pair (p, q) is also distinguishable & mark it as '✗'. The various can be obtained as shown below:

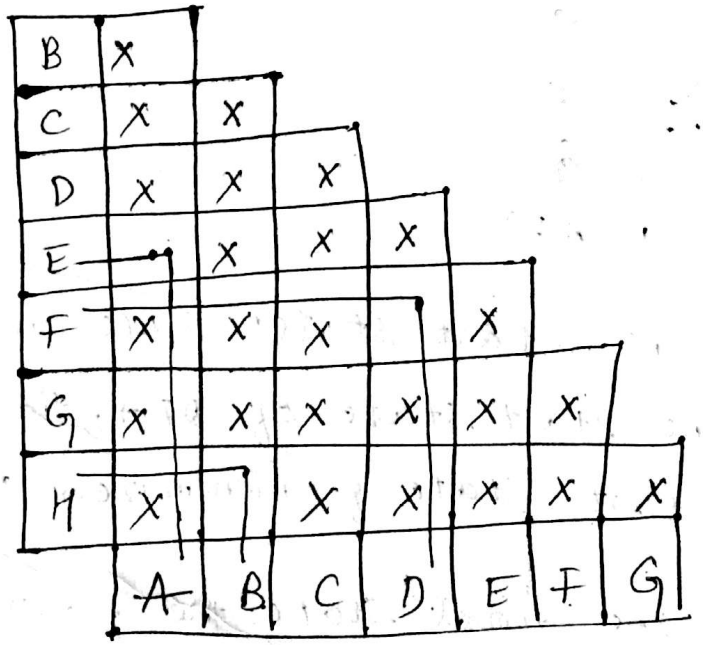
δ	a	b
(A, B)	(B, G)	(F, C)
(A, D)	(B, C)	(F, G)
(A, E)	(B, H)	(F, F)
(A, F)	(B, C)	(F, G)
(A, G)	(B, G)	(F, E)
(A, H)	(B, G)	(F, C)
(B, D)	(G, C)	(C, G)
(B, E)	(G, H)	(C, F)
(B, F)	(G, C)	(C, G)
(B, G)	(G, G)	(C, E)
(B, H)	(G, G)	(C, C)
(D, E)	(C, H)	(G, F)
(D, F)	(C, C)	(G, G)
(D, G)	(C, G)	(G, E)
(D, H)	(C, G)	(G, C)
(E, F)	(H, C)	(F, G)
(E, G)	(H, G)	(F, E)
(E, H)	(H, G)	(F, C)
(F, G)	(C, G)	(G, E)
(F, H)	(C, G)	(G, C)
(G, H)	(G, G)	(E, C)

Step 3: Again consider the pairs (p, q) which are not marked in the above table and see whether the corresponding pairs (r, s) or 0 & 1 are marked as shown below.

	a	b
(p, q)	(r, s)	(r, s)
(A, E)	(B, H)	(F, F)
(A, G) ✓	(B, G) ✓	(F, E) ✓
(B, H)	(G, G)	(C, C)
(D, F)	(C, C)	(G, G)
(E, G) ✓	(H, G) ✓	(F, E) ✓

∴ Mark the pairs (A, G) & (E, G).

i.e Final table is as follows.



∴ Indistinguishable pairs are:
(A, E), (B, H) & (D, F)

Distinguishable pairs:
C & G

Minimizing the DFA:

Step 1: Find the distinguishable & indistinguishable pairs.

D. : (A, C) & G

I : (A,E), (B,H) & (D,F)

Step 2: Obtain the states of minimized DFA:

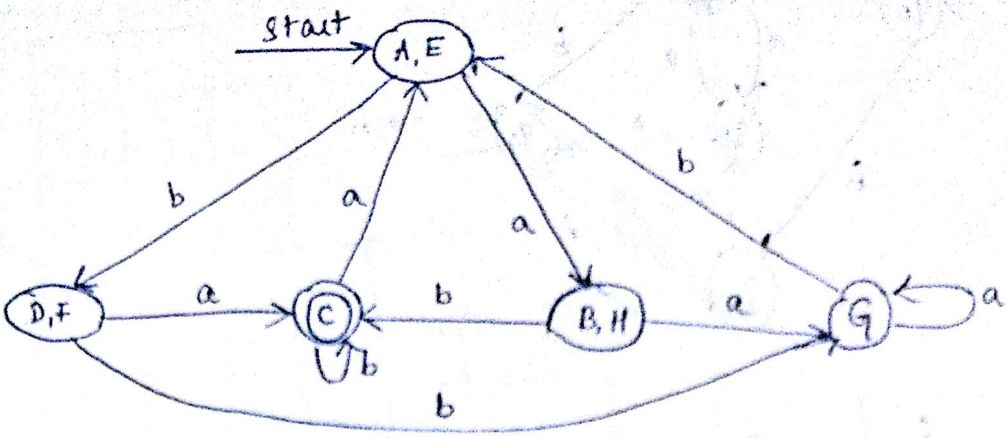
(A,E), (B,H), C, (D,F), G

Step 3: Compute the transition table. Use the transition table of given DFA & obtain the transitions for the minimized DFA as shown below:

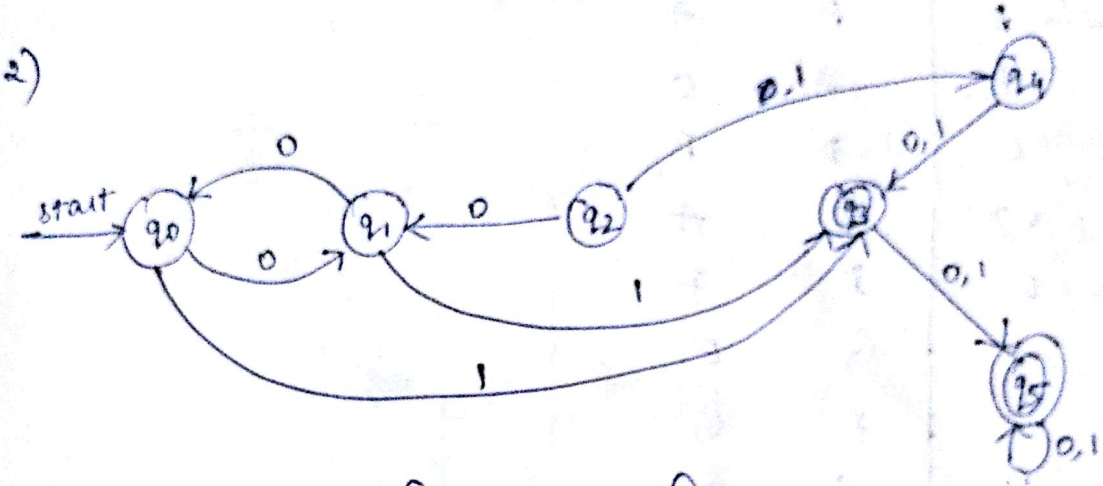
δ	a	b
→ (A,E)	(B,H)	(D,F)
(B,H)	G	C
*C	(A,E)	C
(D,F)	C	G
G	G	(A,E)

Step 4: Identify the start state: Since the group (A,E) has the start state of DFA, the group (A,E) is the start state of minimized DFA.

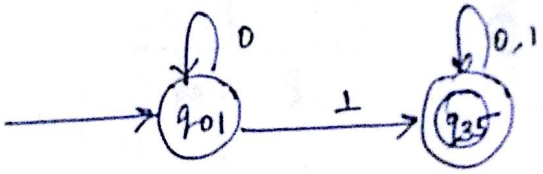
Step 5: Identify the final state: The group C is the final state of given DFA. So, C will be the final state of minimized DFA. The transition diagram of the minimized DFA is shown below:



Ex 2)



Sol^n



Ex 3)

δ	0	1
→ A	B	E
B	C	F
* C	D	H
D	E	H
E	F	I
* F	G	B
G	H	B
H	I	C
* I	A	E

⇒ Indistinguishable pairs

(A,D) (A,G) = (A, D, G) = (D, G)

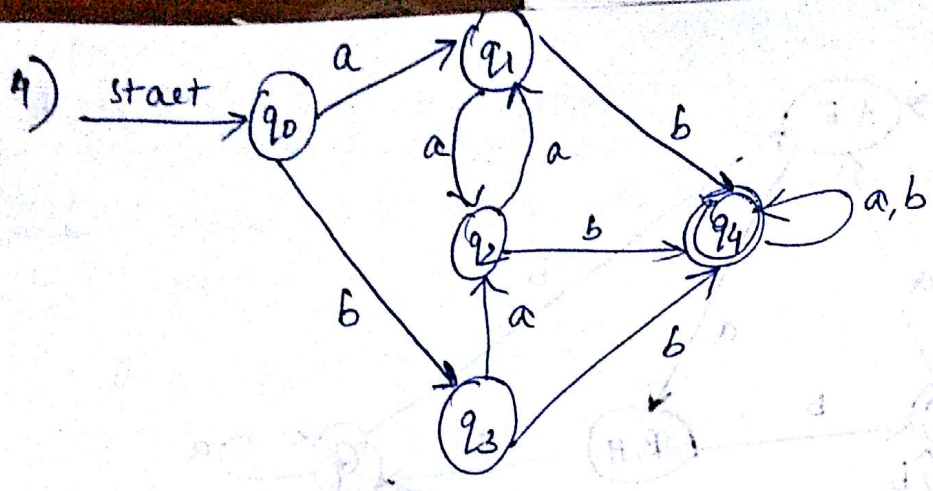
(B,E) (B,H) = (B, E, H) = (E, H)

(C,F), (C,I) = (C, F, I) = (F, I)

Distinguishable pairs does not exist.

TT: δ	a	b
→ (A,D,H)	(B,E,H)	(B,E,H)
(B,E,H)	(C,F,I)	(C,F,I)
* (C,F,I)	(A,D,H)	(B,E,H)

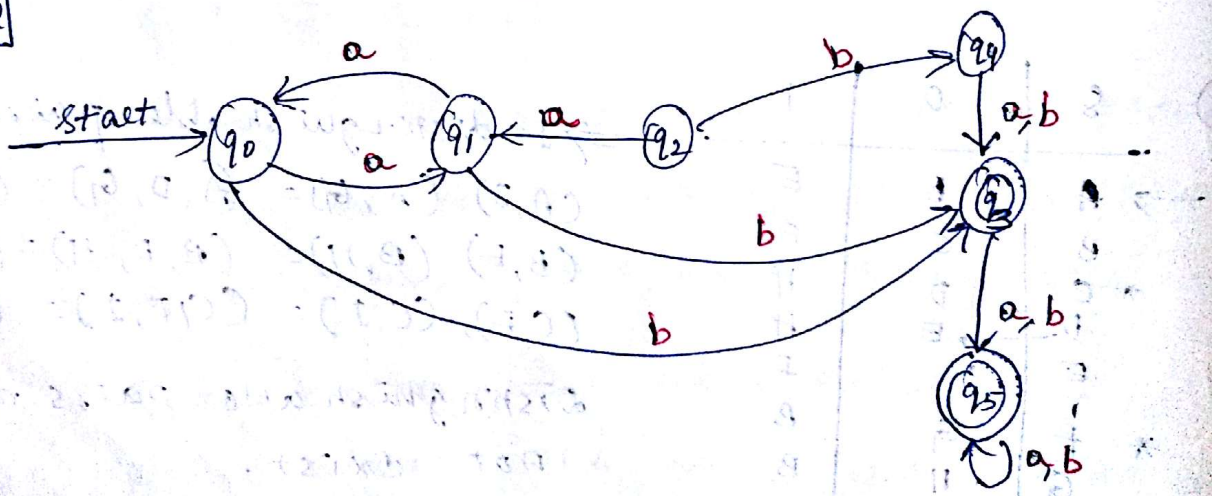
4)



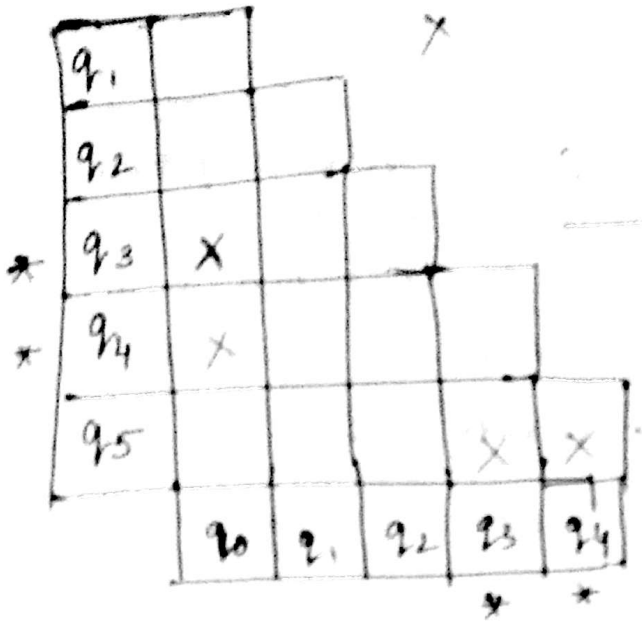
5)

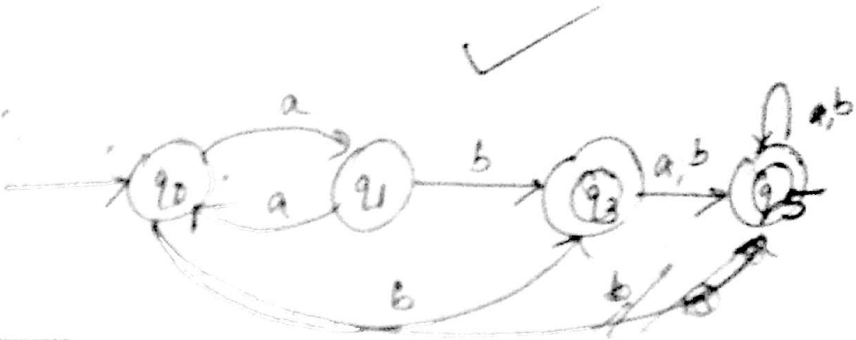
→	δ	0	1
→	A	B	A
	B	A	C
	C	D	B
*	D	D	A
	E	D	F
	F	G	E
	G	F	G
	H	G	D

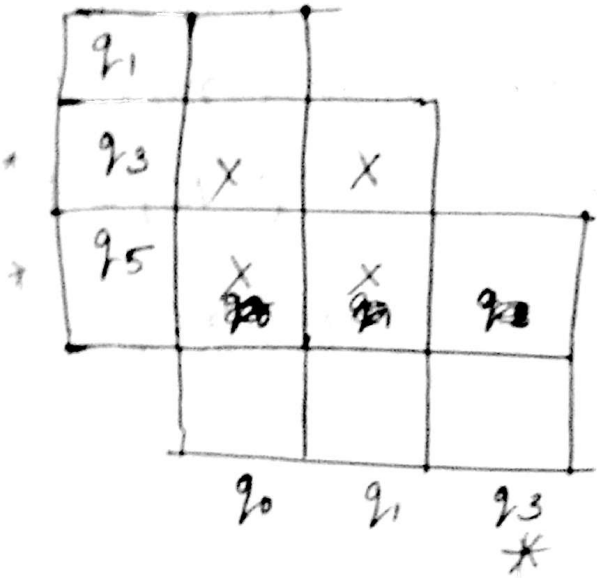
Ex 2



From the above fig., it is clear that states q_3 & q_4 are not reachable from the start state they can be removed.







Indistinguishable pairs are: (q_0, q_1) & (q_3, q_5)

No. Dist pairs

∴ minimized TT is shown below:

states	a	b

Unit II. Regular Expressions:

The language accepted by a DFA, NFA & ε-NFA is called regular language. [i.e a lang. is regular if there exists a finite acceptor for it].

A & regular language can be described using regular expressions consisting of the symbols such as alphabets Σ & the operators, & parentheses.

The three operators used to obtain a regular expression are:

① (+ operator) → Union (least precedence)
② (· or ∘) → Concatenation (next least →)
③ (* or ⁺) → closure (highest →)

Formal def^n of a regular expression:

Let Σ be a given alphabet. Then

1] ϕ, ∧ & a ∈ Σ are all regular exp^n.s. These are called primitive R.E

2] If r_1 & r_2 are R.Es, so are $r_1 + r_2$, $r_1 \cdot r_2$, r_1^* & (r1)

3] A string is a regular expression if & only if it can be derived from the primitive R.Es by a finite no of applications of the rules in (2).

Languages associated with REs

R.Es can be used to describe some simple languages. If r is a R.E, we will let L(r) denote the language associated with r. This is defined as follows:

The lang. $L(\gamma)$ denoted by any RE γ is defined by the following rules.

1) ϕ is a RE denoting the empty set.
2) ε is a R.E denoting $\{\lambda\}$
3) for every $a \in \Sigma$, a is a RE denoting $\{a\}$.

If γ_1 & γ_2 are REs, then

4) $L(\gamma_1 + \gamma_2) = L(\gamma_1) \cup L(\gamma_2)$
5) $L.(\gamma_1 . \gamma_2) = L(\gamma_1).L(\gamma_2)$
6) $L((\gamma_1)) = L(\gamma_1)$
7) $L(\gamma_1^*) = (L(\gamma_1))^*$

} These rules are used to reduce $L(\gamma)$ to simpler components recursively.

To see what language a given expression denotes, we apply these rules repeatedly.

R.E	Meaning
1) a^*	string consisting of any no of a's (or string consisting of zero or more a's)
2) a^+	string consisting of atleast one a (or string consisting of one or more a's).
3) $(a+b)$	string consting of either one a or one b
4) $(a+b)^*$	set of strings of a's & b's of any length including the NULL string.
5) $(a+b)^* abb$	ending with the string abb
6) $ab(a+b)^*$	starting with ab

7) $(a+b)^* aa (a+b)^*$ ⇒ having a substring a

8) $\{ a^* b^* c^* \}$ ⇒ set of strings of any no of a's (
string also), followed by any no of b's
c's

9) $a^+ b^+ c^+$ → atleast one a followed by str
of atleast one b followed by string cons
atleast one c.

10) $aa^* bb^* cc^*$ ⇒ set of strings consisting
one a

11) $(a+b)^* (a+bb)$ ⇒ set of strings of a's & b's
with either a or bb.

12) $(aa)^* (bb)^* b$ ⇒ set of strings consisting
followed by odd no of b's.

13) $(0+1)^* 000$ ⇒ set of strings of 0's & 1's endi
consecutive zero's (or ending wi

14) $(11)^*$ ⇒ consisting of even no of b's.

15) $01^* + 1$ ⇒ the language represented is the s
string consisting of a zero followed
possibly including none.

16) $(01)^* + 1$ ⇒ The lang. consisting of a stri
of (01)'s that repeat zero or m

17) $0 (1^* + 1)$ ⇒ set of strings consisting a z
any no of 1's.

18) $(1+\varepsilon)(00^* 1)^* 0^*$ ⇒ strings of 0's & 1's w
consecutive 1's.