13/1/25

ASSIGNMENT 1

Unit - 1 & 2 Introduction to Compiler & lexical Analysis

Q.1 Explain different phases of compiler.

→ The process of compilation is divided into several phases, each of which performs a specific function to translate high-level source code to executable machine code.

1. Analysis Phase

(i) lexical Analysis (Scanning)

It converts the source code into tokens, removing comments and whitespace. It identifies keywords, operators and punctuation, grouping characters into meaningful lexemes. Tokens are generated for these lexemes.

eg: int a=5; → Tokens: int, a, =, 5, ;

(ii) Syntax Analysis (Parsing)

It checks the grammatical structure of tokens using context-free grammar. It outputs a parse tree, verifying if the tokens form a valid statement.

eg: int a=5; → Valid Declaration.

(iii) Semantic Analysis

It ensures the program's meaning is correct. It checks type compatibility, resolves identifiers and enforces language rules.

eg: int a = "hello"; → Error: type mismatch

*Diagram of Phases of Compiler

(iii) Intermediate Code Generation.

The Syntax Tree is transformed into an intermediate expression (IR) that is platform-independent. This simplifies optimization and further code generation.

eg: a = b + c; → t1 = b + c, a = t1.

2. Synthesis Phase

(i) Code Optimization

It improves the intermediate code by eliminating redundancies and simplifying expressions to enhance performance.

eg: Before : t1 = b + c Optimized : t2 = b + c + d
t2 = t1 + d

(ii) Code Generation

Intermediate code is converted into machine or assembly code specific to the target platform. Memory is allocated and control flow is translated into machine instructions

eg: t2 = b + c + d → LOAD R1, b, ADD R1, c, STORE R1, t2.

Q.2 Explain Semantic Analysis and Syntax Analysis phases of compiler with suitable example. Also explain the reporting errors by these two phases.

-4 Definition is same as Question-1, for both semantic and syntax analyses.

* Error reporting :-

1. Syntax Analysis

① Unmatched Parentheses :

```
if(x > y { z = x + y ; }
```

Error : Excep Expected ')' after 'y'

② Missing Semicolon.

```
z = x + y
```

Error : Missing semicolon at the end of the statement

2. Semantic Analysis

① Type Mismatch :

```
int a = "hello" ;
```

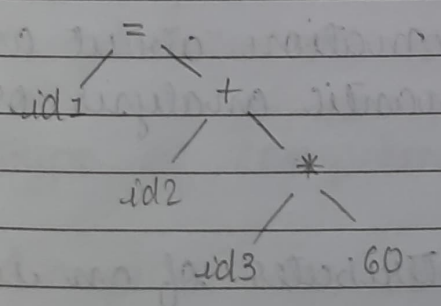
Error : Cannot assign 'String' to 'int'

② Undeclared Variable :

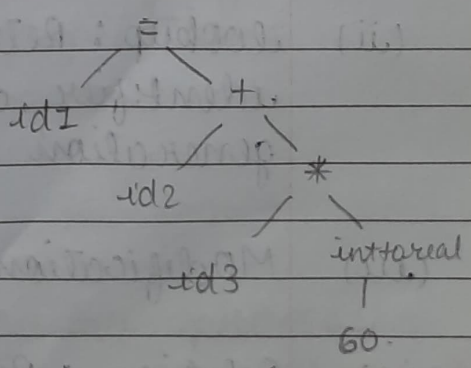
```
x = 5 ;    (in C)
```

Error : Variable 'x' is not declared

Syntax Analysis



Semantic Analysis



Q.3 Write a short note on Symbol Table Management.

—↓ The symbol table is a crucial data structure used in a compiler to store and manage information about identifiers in a source program (such as name, type, scope and memory location). It enables fast lookups during compilation and ensures consistency by checking for duplicates and conflicts in scope and type.

- Structure of a Symbol Table :-

It can be implemented using data structures like hash tables, trees or linked lists, depending on performance requirements. It typically stores symbol name, type, scope, memory address, additional attributes and status.

- Symbol Table Operations :-

(i) Insertion : Add a new identifier when it is declared in the source code.

(ii) Lookup : Retrieve information about an identifier during semantic analysis or code generation

(iii) Modification : Update attributes of an identifier.

(iv) Deletion : Remove identifiers that go out of scope.

- Types of Symbol Tables :-

(i) local (ii) Global (iii) Nested.

eg:
```
int x;
float y;
x = 1.0;
y = x + 2.5;
```

- During the compilation, Symbol Table will store:
  - x : Type = integer, Scope = global
  - y : Type = float, Scope = global
  - After usage : Symbol table will still have 'x' and 'y' marked as initialized with their corresponding values.

Q.4 What is the pass of a compiler? Explain how the single and multi-pass compilers work.

→ A Pass refers to a single traversal through the source code by the compiler, performing a specific task or set of tasks. A compiler may process the source code in one or more passes. Each pass typically focuses on a particular phase of compilation.

- Type of compiler ~~Pb~~ Passes :-

(i) Single - Pass Compiler.

A Single Pass compiler processes the source code in just one pass through

eg: An interpreter or a simple compiler for a language like Pascal or Fortran in its early versions, where each line is compiled and executed sequentially.

the code. During this pass, the compiler performs all the necessary tasks and directly produces the output.

This approach is efficient in terms of time but it has limitations in terms of optimization and handling complex language constructs.

(ii) Multi-Pass Compiler.

A Multi-Pass compiler performs multiple passes over the source code. Each pass focuses on a specific task, such as parsing, semantic analysis, optimization or code generation. In Multi-pass compilers, intermediate code is often generated during earlier passes and later passes optimize and refine it.

The multi-pass approach is more flexible and allows for optimizations that a single-pass compiler cannot achieve but it may requires more time and resources.

eg: Modern compilers like GCC (GNU compiler Collection) and JVM compilers use multiple passes for better optimization and error checking.

Q.5 List out phases of compiler. Write a brief note on lexical Analyzer.

→ A compiler is a complex program that translates a high-level source code written in a programming language into machine code. (Phases are same as Question-1)

→ Lexical Analyzer, also known as the scanner is the phase of compiler. It reads the source code character and groups them into meaningful sequences called lexemes, which are represented as tokens.

• Functions of lexical Analyzer are :-

(i) Tokenization : Breaks the input into tokens, which are atomic units such as keywords, identifiers, operators and symbol.

(ii) Removing whitespace and comments : Eliminates unnecessary whitespace, tabs and comments to simplify subsequent phases.

(iii) Error ~~&~~ Detection : Identifies invalid characters, unrecognized symbols or incomplete tokens.

(iv) Symbol Table Interaction : Updates the symbol table with information about identifiers.

eg: Input : int num = 42;

① Break the code into lexemes : int, num, =, 42, ;.

② Generate Tokens :

<keyword, int>, <identifier, num>, <operator, =>, <number, 42>, <delimiter, ;>

→ It provides efficiency, error-detection and it is language-specific

Q.6 How do the parser and scanner communicate? Explain with the block diagram communication between them. Also explain: what is input buffering?

→ The Scanner and the Parser work together in sequence to process the source code. The scanner is responsible for converting the raw source code into tokens, which are the basic building blocks of the language. Then the parser builds a parse tree or AST based on the language's grammar.

Communication Process :-

i) Scanner's Role

The scanner reads the source code, (character by character) identifies patterns and converts them into tokens. It then sends these tokens to the parser, one at a time.

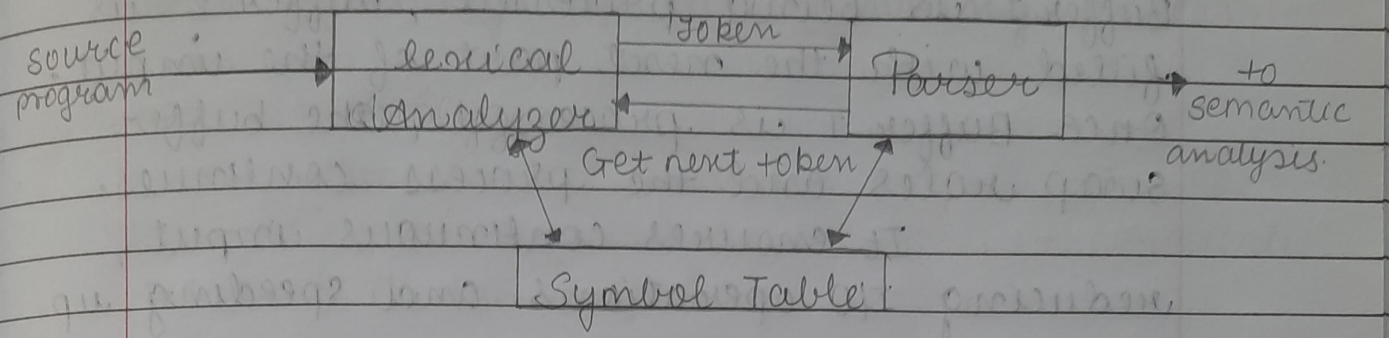
ii) Parser's Role.

The parser receives tokens from the scanner and checks if the sequence follows the syntax rules of the language. It constructs a syntax tree (parse) by matching tokens to grammar rules.

iii) Communication

The scanner sends token to the parser one by one. If the parser needs more tokens, it requests them from the

scanner. This continues until the entire code is parsed.



→ Input buffering is a technique used by the scanner to improve efficiency by storing portions of the source code in memory, reducing the need for frequent disk accesses. It allows the scanner to process the input one character or lexeme at a time, while also enabling lookahead to identify multi-character lexemes. This reduces I/O overhead and supports more effective tokenization.

Q.7 write two methods used in lexical analyzer for buffering the input. Which technique is used for speeding up the lexical analyzer?

→ The mainly two techniques for input buffering :-

1. Buffer Pairs

Also called double buffering, it uses two buffers to hold portions of input data, allowing the scanner to process one buffer while the other is being filled with the next chunk of the input, improving lexical analysis speed.

Working is like the input is split into Buffer 1 and Buffer 2. While Buffer 1 is being processed, Buffer 2 is filled with the next part of the input. Once Buffer 1 is processed, the buffer swap roles and the process continues.

It ensures continuous input, reducing I/O operations and speeding up the scanning process.

eg: The Source code input :

```
int x = 10 + 5;   ← Buffer 1 might hold
int y = 20 - 3;   ← Buffer 2 might hold
```

While the lexical analyzer processes the contents of Buffer 1, Buffer 2 is loaded with the next part of the code.

2. Sentinels.

Sentinels are special markers placed at the end of the input buffer to signal the end of the data, helping the scanner know where to stop processing.

Working is like a marker is placed at the end of the input string, so the scanner knows when to stop processing and avoid accessing memory outside the allocated buffer. It also allows the scanner to avoid boundary checks when reading characters, making it simpler and faster to read the input.

It simplifies the code and improve efficiency by eliminating boundary checks and speeding up the scanning process.

eg: int x = 10+5;

The lexical analyzer might place a sentinel ('$$' or '\0') at the end to indicate the end of the input.

→ Buffer Pairs is most commonly used for speeding up the lexical analyzer.

Q.8 Write a brief on input buffering techniques OR Explain buffer pairs and Sentinels OR Write short note on input buffering.

→ Same answer given in Question-7.

Q.9 Define the following terms:

1) Token

→ A token is a categorized unit of the input source code, representing a sequence of characters that form a meaningful unit for the compiler. Tokens are the output of the lexical ~~analyzer~~ analysis phase and are passed to the parser for further processing.

eg: The source code, int x = 10; , the tokens are: <keyword, int>, <identifier, x>, <operator, =>, <number, 10>, <delimiter, ;>

2) Pattern

—> A Pattern is a regular expression or set of rules that define the structure of a token. It is a description of the form that a token can take in the source code. Patterns help the lexical analyzer identify and classify sequences of characters as tokens.

eg:
- The pattern for an identifier can be: ^[a-zA-Z_][a-zA-Z0-9_]*$; which means it can begin with a letter or an underscore and can be followed by letters, digits or underscore.
- The Pattern for a number could be: ^[0-9]+$; which means a number consists of one or more digits.

3. lexeme.

—> A lexeme is an actual sequence of characters in the source code that matches a pattern for a token. It represents the real string of characters that the lexical analyzer scans and categorizes as particular token.

eg: The source code, int x = 10; , the lexemes are:
- int → corresponds to the token <keyword, int>
- x → <identifier, x>
- = → <operator, =>
- 10 → <number, 10>
- ; → <delimeter, ;>

The lexeme as the actual substring from the source code that matches the defined pattern.

Q.10 Define lexeme, token and pattern. Identify the lexemes that make up the tokens in the following program segment. Indicate corresponding token and pattern.

(1)
```
void swap (int a, int b) {
    int k;
    k = a;
    a = b;
    b = k;
}
```

*EM = Exact Match.

Lexeme	Token	Pattern
void	Keyword	EM(void)
swap	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
(	Punctuation	EM(()
int	Keyword	EM(int)
a	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
,	Punctuation	EM(,)
int	Keyword	EM(int)
b	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
)	Punctuation	EM())
{	Punctuation	EM({)
int	Keyword	EM(int)
k	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
;	Punctuation	EM(;)
k	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
=	Operator	EM(=)
a	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
;	Punctuation	EM(;)
a	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
=	Operator	EM(=)
b	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
;	Punctuation	EM(;)

b	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
=	Operator	EM(=)
K	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
;	Punctuation	EM(;)
}	Punctuation	EM(})

(2)

```
String message = "Welcome to Java!";
System.out.println(message);
```

String	Keyword	EM(String)
message	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
=	Operator	EM(=)
"welcome to Java!"	String	"[^"]*"
System	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
.	Punctuation	EM(.)
out	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
.	Punctuation	EM(.)
println	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
(	Punctuation	EM(()
message	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
)	Punctuation	EM())
;	Punctuation	EM(;)

(3)

```
#include <stdio.h>
int main() {
    printf("Hello, C! \n");
    return 0;
}
```

#include	Preprocessor Directive	EM(#include)
<stdio.h>	Header File	EM(<stdio.h>)
int	Keyword	EM(int)
main	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
(	Punctuation	EM(()

)	Punctuation	EM())
{	Punctuation	EM({)
printf	Identifier	^[a-zA-Z_][a-zA-Z0-9_]*$
(	Punctuation	EM(()
"Hello, C! \n"	String	"[^"]*"
)	Punctuation	EM())
;	Punctuation	EM(;)
return	Keyword	EM(Return)
0	Integer	^[0-9]+$
;	Punctuation	EM(;)
}	Punctuation	EM(})

Q.11 Find the regular expression corresponding to given statement, subset of {0,1}*

i) The language of all strings containing at least one 0 and at least one 1.

→ String : 01, 10

R.E : (0+1)* 0 (0+1)* 1 (0+1)*

ii) The language of all strings containing 0's and 1's both are even.

→ String : ~~00~~, ~~11~~, 0011, 000011

R.E : ((00|11)|(01|10)(00|11)*(01|10))*

iii) The language of all strings containing at most one pair of consecutive 1's.

→ String : 0, 01, 0001, 1001

R.E : (0|1)*11?(0|1)*

iv) The language of all strings that do not end with 01.

—4 string : 0, 1, 101, 110
R.E: (1|0)*(0|1|11|00)

Q.12 What are regular expression ? Find the regular expression described by DFA {{A, B}, {0, 1}, δ, A, {B}}, where δ is detailed in following table.

	0	1
A	A	B
B	ϕ	A

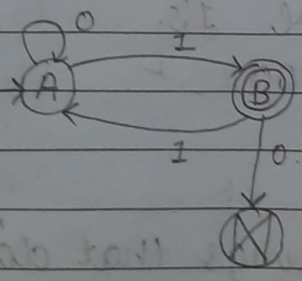
Please note B is accepting state. Describe the language defined by the regular expression.

—4 A Regular Expression (RE) is a sequence of characters that define a search pattern. It is used to match strings or sets of strings in text processing. They are a formal way to describe the patterns of strings that belong to a specific language

Here, States Q = {A, B}
Σ = {0, 1}
q_0 = A
f = {B}

δ	0	1
A	A	B
B	ϕ	A

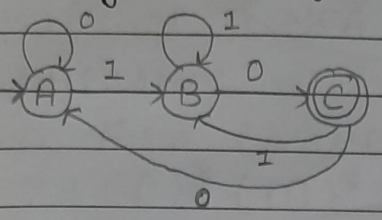
So, the DFA would be :



So, the regex would be,
0*1(1|10*1)*

Q.13 Draw Deterministic Finite Automata for the binary strings ending with 10.

—4



δ	0	1
A	A	B
B	C	B
C	A	B

Here, $q_0 = A$

$Q = \{A, B, C\}$

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

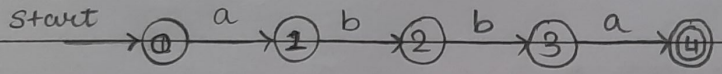
0010010 - Accepted

1001 - Rejected

Q.14 Convert the following regular expression to NFA:

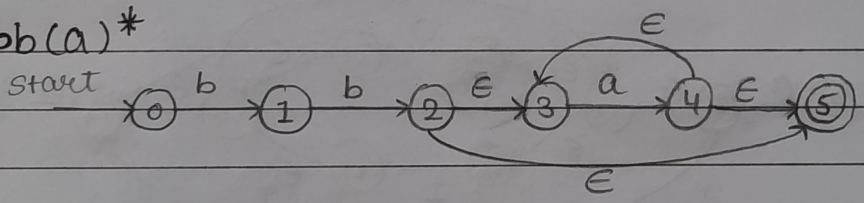
(i) abba

—4



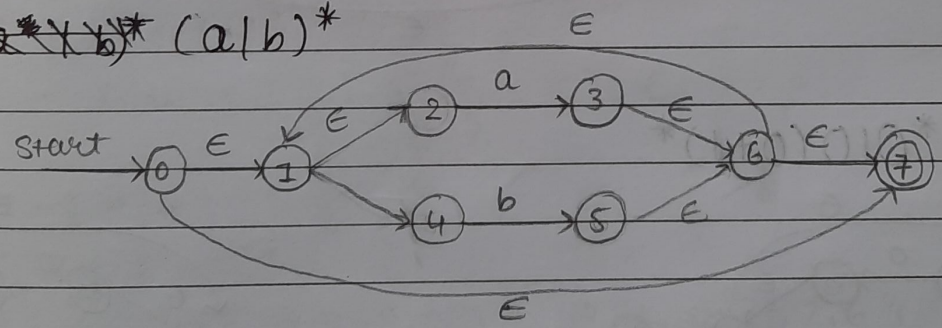
(ii) bb(a)*

—4



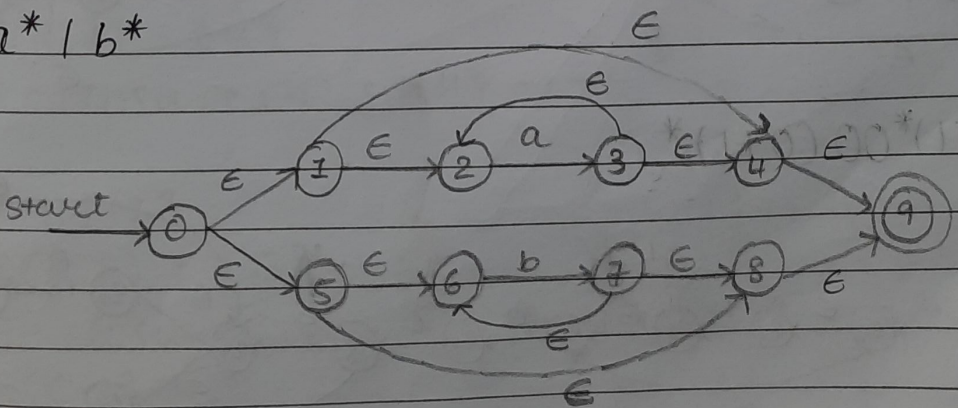
(iii) ~~(a*Xb)*~~ (a|b)*

—4



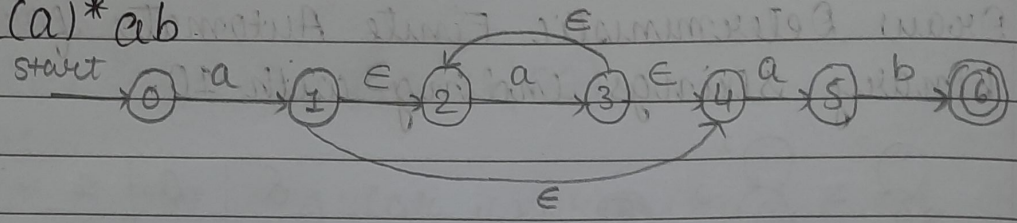
(iv) a* | b*

—4



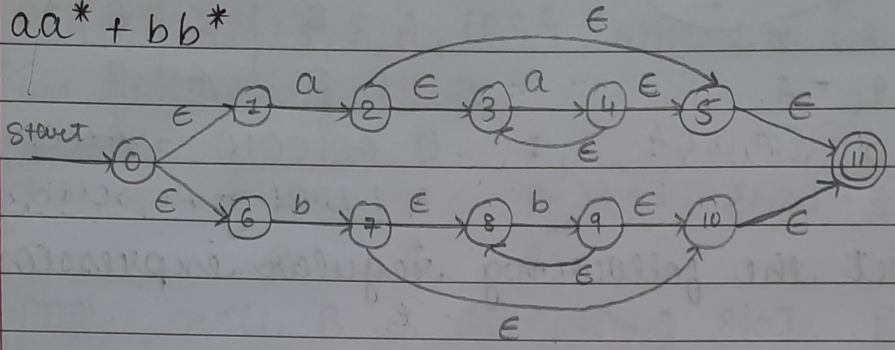
(v) $a(a)^*ab$

→



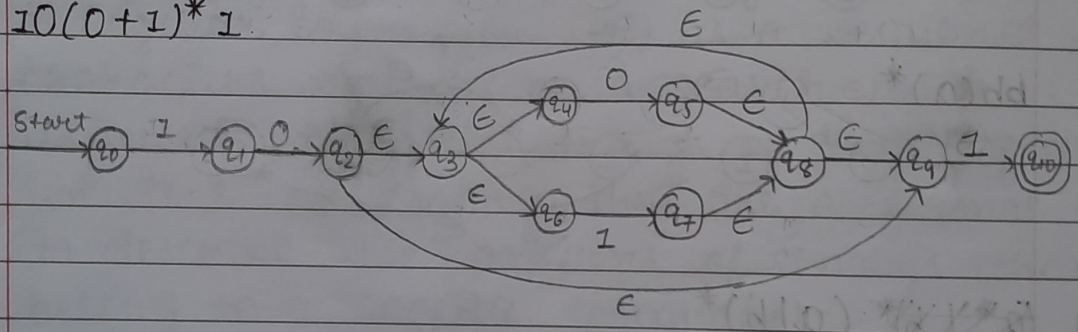
(vi) $aa^* + bb^*$

→



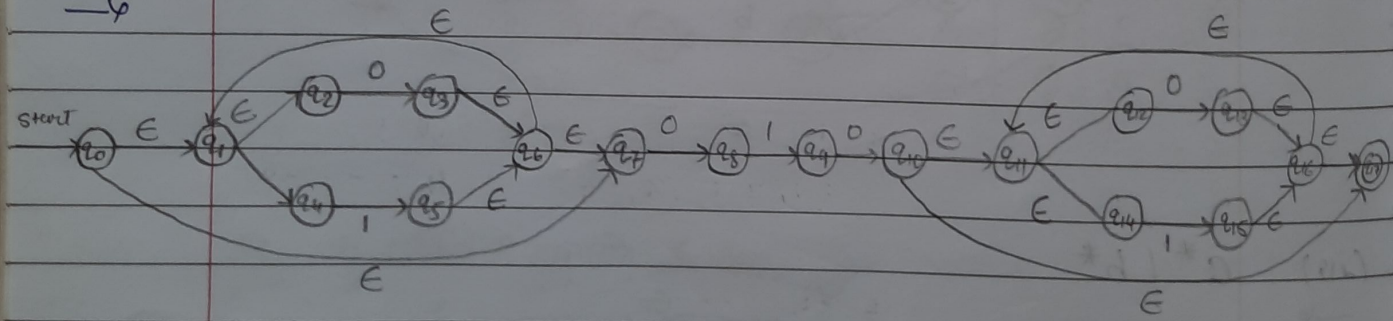
(vii) $10(0+1)^*1$

→



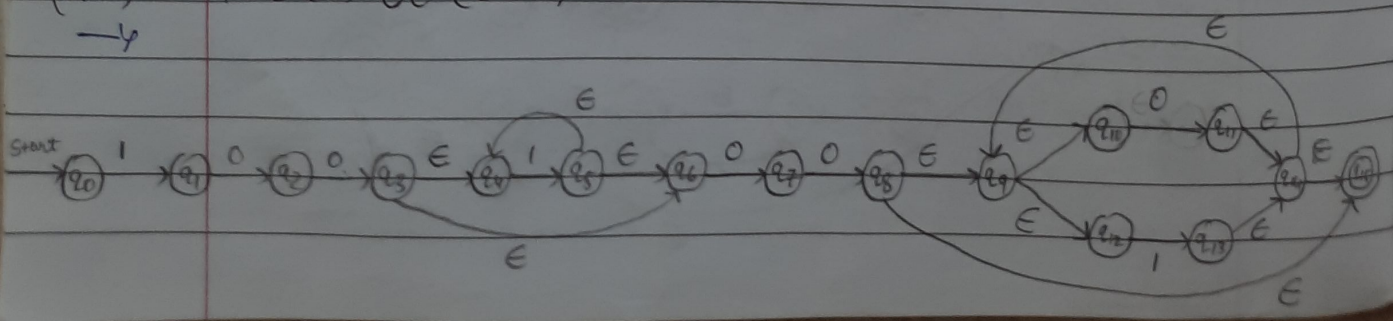
(viii) $(0+1)^*010(0+1)^*$

→



(ix) $100(1)^*00(0+1)^*$

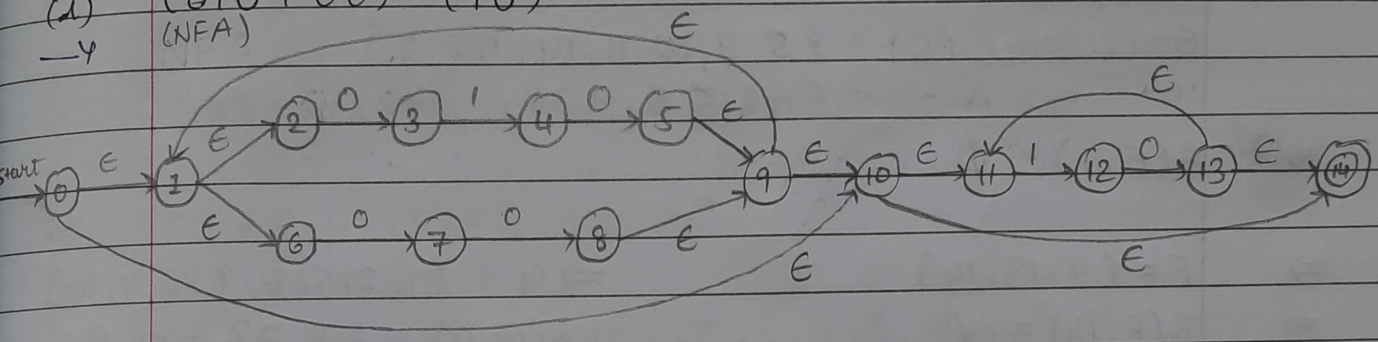
→



Q.15 Convert ~~the~~ following regular expression to DFA using subset construction Method. OR construct a NFA for the following regular expression using Thompson's notation and then convert it into DFA.

(i) $(01O+00)^*(10)^*$

—y (NFA)



- ε-closure(0) = {0, 1, 2, 6, 10, 11, 14} --- A.

⇒ A = {0, 1, 2, 6, 10, 11, 14}

- δ(A, 0) = {3, 7}
  ε-closure(3, 7) = {3, 7} --- B.
- δ(A, 1) = {12}
  ε-closure(12) = {12} --- C.

⇒ B = {3, 7}

- δ(B, 0) = {8}
  ε-closure(8) = {8, 9, 10, 11, 14, 1, 2, 6}
  = {1, 2, 6, 8, 9, 10, 11, 14} --- D
- δ(B, 1) = {4}
  ε-closure(4) = {4} --- E

⇒ C = {12}

- δ(C, 0) = {13}
  ε-closure(13) = {11, 13, 14} --- F
- δ(C, 1) = ϕ

$\Rightarrow$ $D = \{1,2,6,8,9,10,11,14\}$

- $\delta(D,0) = \{3,7\}$ --- B
- $\delta(D,1) = \{12\}$ --- C.

$\Rightarrow$ $E = \{4\}$

- $\delta(E,0) = \{5\}$

ϵ-closure $(5) = \{5,9,10,11,14,1,2,6\}$
$= \{1,2,5,6,9,10,11,14\}$ --- G

- $\delta(E,1) = \phi$

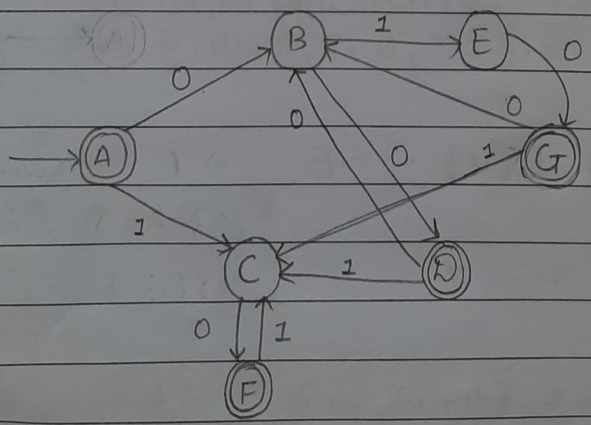
$\Rightarrow$ $F = \{11,13,14\}$

- $\delta(F,0) = \phi$
- $\delta(F,1) = \{13\}$ --- C

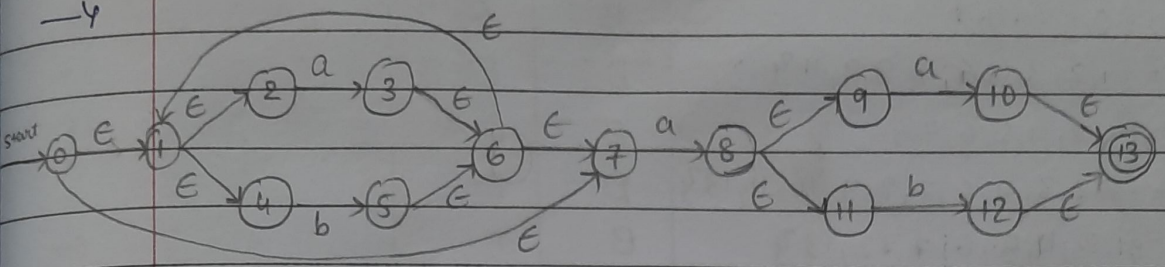
$\Rightarrow G = \{1,2,5,6,9,10,11,14\}$

- $\delta(G,0) = \{3,7\}$ --- B
- $\delta(G,1) = \{12\}$ --- C

States	0	1
$A^* = \{0,1,2,6,10,11,14\}$	B	C
$B = \{3,7\}$	D	E
$C = \{12\}$	F	ϕ
$D^* = \{1,2,6,8,9,10,11,14\}$	B	C
$E = \{4\}$	G	ϕ
$F^* = \{11,13,14\}$	ϕ	C
$G^* = \{1,2,5,6,9,10,11,14\}$	B	C



(iii) $(a+b)^* a(a+b)$



- ϵ-closure (0) = {0, 1, 2, 4, 7} --- A

⇒ A = {0, 1, 2, 4, 7}
- δ(A, a) = {3, 8}
  ϵ-closure (3, 8) = {1, 2, 3, 4, 6, 7, 8, 9, 11} --- B
- δ(A, b) = {5}
  ϵ-closure (5) = {1, 2, 4, 5, 6, 7} --- C

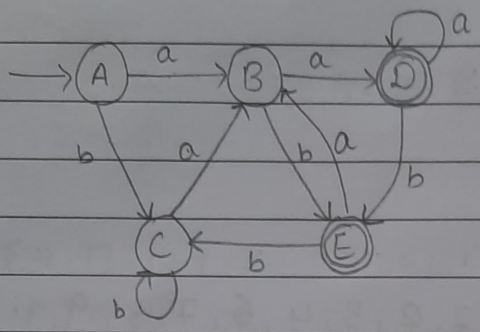
⇒ B = {1, 2, 3, 4, 6, 7, 8, 9, 11}
- δ(B, a) = {3, 8, 10}
  ϵ-closure (3, 8, 10) = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13} --- D
- δ(B, b) = {5, 12}
  ϵ-closure (5, 12) = {1, 2, 4, 5, 6, 7, 12, 13} --- E

⇒ C = {1, 2, 4, 5, 6, 7}
- δ(C, a) = {3, 8} --- B
- δ(C, b) = {5} --- C

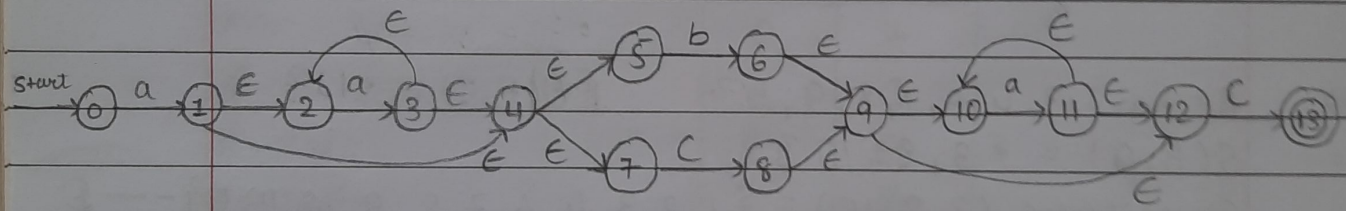
⇒ D = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13}
- δ(D, a) = {3, 8, 10} --- D
- δ(D, b) = {5, 12} --- E

⇒ E = {1, 2, 4, 5, 6, 7, 12, 13}
- δ(E, a) = {3, 8} --- B
- δ(E, b) = {5} --- C

States	a	b
A = {0,1,2,4,7}	B	C
B = {1,2,3,4,6,7,8,9,11}	D	E
C = {1,2,4,5,6,7}	B	C
D* = {1,2,3,4,6,7,8,9,10,11,13}	D	E
E* = {1,2,4,5,6,7,12,13}	B	C



(iii) $aa^*(b|c)a^*c\#$



- ϵ-closure(0) = {0} --- A

⇒ A = {0}

- $\delta(A,a) = \{1\}$

  ϵ-closure(1) = {1,2,4,5,7} --- B

- $\delta(A,b) = \phi$ $\delta(A,c) = \phi$

⇒ B = {1,2,4,5,7}

- $\delta(B,a) = \{3\}$

  ϵ-closure(3) = {2,3,4,5,7} --- C

- $\delta(B,b) = \{6\}$

  ϵ-closure(6) = {6,9,10,12} --- D

- $\delta(B,c) = \{8\}$

  ϵ-closure(8) = {8,9,10,12} --- E

⇒ C = {2, 3, 4, 5, 7}
- δ(C, a) = {3} --- C
- δ(C, b) = {6} --- D
- δ(C, c) = {8} --- E

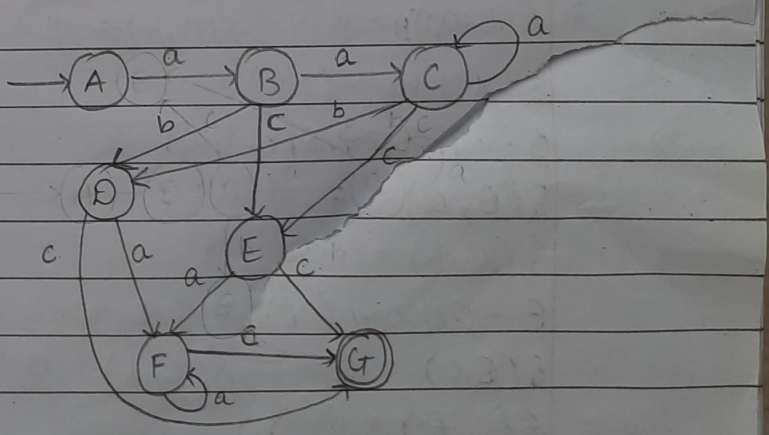
⇒ D = {6, 9, 10, 12}
- δ(D, a) = {11}
  ε-closure(11) = {10, 11, 12} --- F
- δ(D, b) = ϕ
- δ(D, c) = {13}
  ε-closure(13) = {13} --- G

⇒ E = {8, 9, 10, 12}
- δ(E, a) = {11} --- F
- δ(E, b) = ϕ
- δ(E, c) = {13} --- G

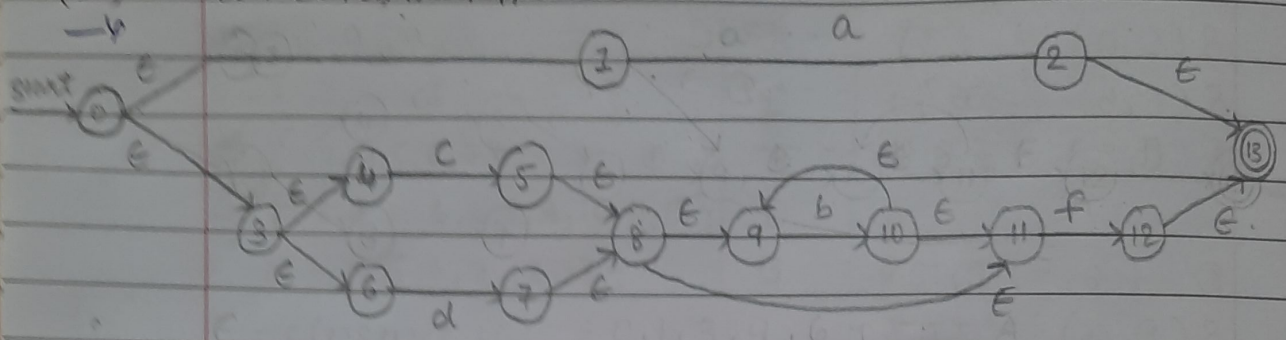
⇒ F = {10, 11, 12}
- δ(F, a) = {11} --- F
- δ(F, b) = ϕ
- δ(F, c) = {13} --- G

⇒ G = {13}
- δ(G, a) = ϕ δ(G, b) = ϕ δ(G, c) = ϕ

States	a	b	c
A = {0}	B	ϕ	ϕ
B = {1, 2, 4, 5, 7}	C	D	E
C = {2, 3, 4, 5, 7}	C	D	E
D = {6, 9, 10, 12}	F	ϕ	G
E = {8, 9, 10, 12}	F	ϕ	G
F = {10, 11, 12}	F	ϕ	G
*G = {13}	ϕ	ϕ	ϕ



(20) $a+(c|d)b^*f\#$



- ϵ-closure(0) = {0, 1, 3, 4, 6} --- A

$\Rightarrow$ A = {0, 1, 3, 4, 6}

- $\delta(A, a) = \{2\}$

  ϵ-closure(2) = {2, 13} --- B
- $\delta(A, b) = \phi$
- $\delta(A, c) = \{5\}$

  ϵ-closure(5) = {5, 8, 9, 11} --- C
- $\delta(A, d) = \{7\}$

  ϵ-closure(7) = {7, 8, 9, 11} --- D
- $\delta(A, f) = \phi$

$\Rightarrow$ B = {2, 13}

- $\delta(B, a) = \phi$ $\delta(B, b) = \phi$ $\delta(B, c) = \phi$ $\delta(B, d) = \phi$ $\delta(B, f) = \phi$

$\Rightarrow$ C = {5, 8, 9, 11}

- $\delta(C, a) \neq \phi$
- $\delta(C, b) = \{10\}$

  ϵ-closure(10) = {9, 10, 11} --- E
- $\delta(C, c) = \phi$ $\delta(C, d) = \phi$
- $\delta(C, f) = \{12\}$

  ϵ-closure(12) = {12, 13} --- F

$\Rightarrow$ D = {7, 8, 9, 11}

- $\delta(D, a) = \phi$
- $\delta(D, b) = \{10\}$ --- E
- $\delta(D, c) = \phi$ $\delta(D, d) = \phi$
- $\delta(D, f) = \{12\}$ --- F

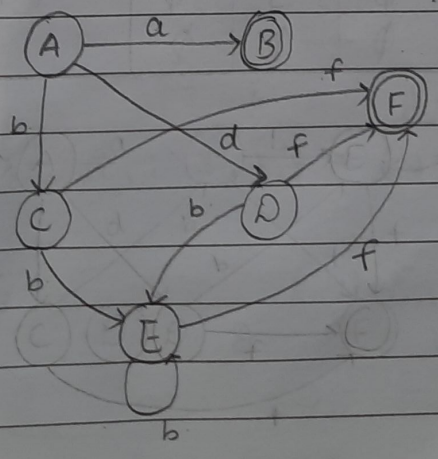
$\Rightarrow$ $E = \{9, 10, 11\}$

- $\delta(E, a) = \phi$
- $\delta(E, b) = \{10\}$ --- E
- $\delta(E, c) = \phi$ $\delta(E, d) = \phi$ $\delta(E, f) = \{12\}$ --- F

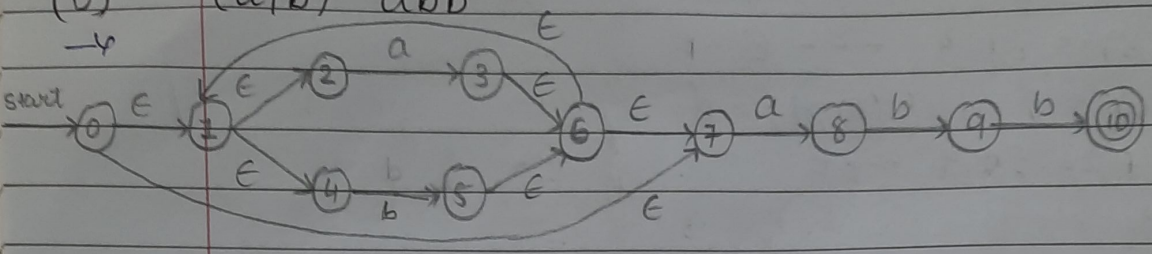
$\Rightarrow$ $F = \{12\}$

- $\delta(F, a) = \phi$ $\delta(F, b) = \phi$ $\delta(F, c) = \phi$ $\delta(F, d) = \phi$ $\delta(F, f) = \phi$

States	a	b	c	d	f
$A = \{0, 1, 3, 4, 6\}$	B	ϕ	C	D	ϕ
$B^* = \{2, 13\}$	ϕ	ϕ	ϕ	d	ϕ
$C = \{5, 8, 9, 11\}$	ϕ	E	ϕ	ϕ	F
$D = \{7, 8, 9, 11\}$	ϕ	E	ϕ	ϕ	F
$E = \{9, 10, 11\}$	ϕ	E	ϕ	ϕ	F
$F^* = \{12, 13\}$	ϕ	ϕ	ϕ	ϕ	ϕ



(iv) $(a|b)^* abb$



- ϵ-closure (0) = {0, 1, 2, 4, 7} --- A

⇒ A = {0, 1, 2, 4, 7}

- $\delta(A, a)$ = {3, 8}
  ϵ-closure (3, 8) = {1, 2, 3, 4, 6, 7, 8} --- B
- $\delta(A, b)$ = {5}
  ϵ-closure (5) = {1, 2, 4, 5, 6, 7} --- C

⇒ B = {1, 2, 3, 4, 6, 7, 8}

- $\delta(B, a)$ = {3, 8} --- B
- $\delta(B, b)$ = {5, 9}
  ϵ-closure (5, 9) = {1, 2, 4, 5, 6, 7, 9} --- D

⇒ C = {1, 2, 4, 5, 6, 7}

- $\delta(C, a)$ = {3, 8} --- B
- $\delta(C, b)$ = {5} --- C

⇒ D = {1, 2, 4, 5, 6, 7, 9}

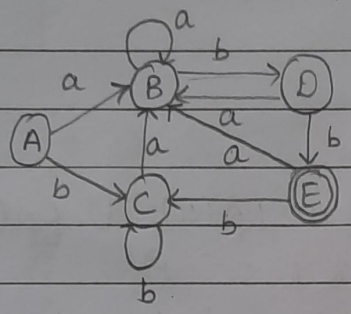
- $\delta(D, a)$ = {3, 8} --- B
- $\delta(D, b)$ = {5, 10}
  ϵ-closure (5, 10) = {1, 2, 4, 5, 6, 7, 10} --- E

⇒ E = {1, 2, 4, 5, 6, 7, 10}

- $\delta(E, a)$ = {3, 8} --- B
- $\delta(E, b)$ = {5} --- C

States	a	b
A = {0,1,2,4,7}	B	C
B = {1,2,3,4,6,7,8}	B	D
C = {1,2,4,5,6,7}	B	C
D = {1,2,4,5,6,7,9}	B	E
E* = {1,2,4,5,6,7,10}	B	C

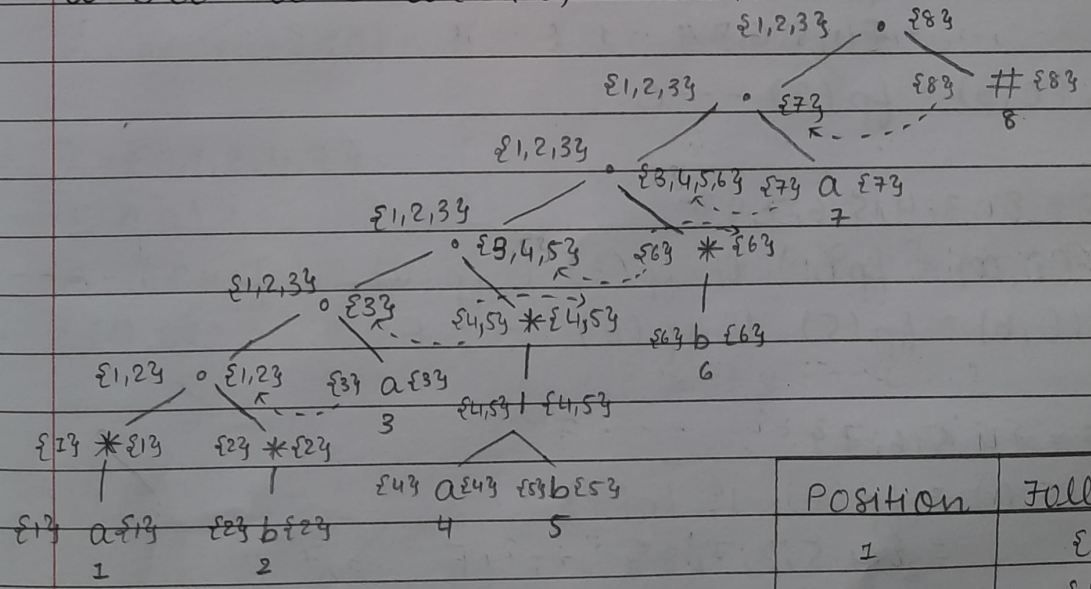
DFA:



Q.16 Construct a DFA without constructing NFA for following regular expression. Find minimized DFA.

$a^*ba^*\ a^*b^*a(a|b)^*\ b^*a\#$

—4



Position	Followups
1	{1,3}
2	{2,3}
3	{4,5,6,7}
4	{4,5,6,7}
5	{4,5,6,7}
6	{6,7}
7	{8}
8	—

- Initial state = {1, 2, 3} --- A

⇒ A = {1, 2, 3}

- δ(A, a) = followpos(1) ∪ followpos(3)
  = {1, 3, 4, 5, 6, 7} --- B
- δ(A, b) = followpos(2)
  = {2, 3} --- C

⇒ B = {1, 3, 4, 5, 6, 7}

- δ(B, a) = fp(1) ∪ fp(3) ∪ fp(4) ∪ fp(7)
  = {1, 3, 4, 5, 6, 7, 8} --- D
- δ(B, b) = fp(5) ∪ fp(6)
  = {4, 5, 6, 7} --- E.

⇒ C = {2, 3}

- δ(C, a) = fp(3)
  = {4, 5, 6, 7} --- E
- δ(C, b) = fp(2) --- C.

⇒ D = {1, 3, 4, 5, 6, 7, 8}

- δ(D, a) = fp(1) ∪ fp(3) ∪ fp(4) ∪ fp(7) --- D.
- δ(D, b) = fp(5) ∪ fp(6) --- E

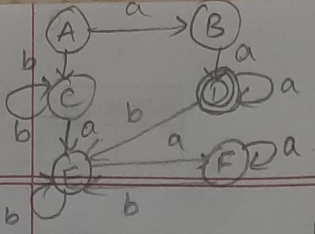
⇒ E = {4, 5, 6, 7}

- δ(E, a) = fp(4) ∪ fp(7)
  = {4, 5, 6, 7, 8} --- F
- δ(E, b) = fp(5) ∪ fp(6) --- E

⇒ F = {4, 5, 6, 7, 8}

- δ(F, a) = fp(4) ∪ fp(7) --- F
- δ(F, b) = fp(5) ∪ fp(6) --- E

DFA:



$\Rightarrow$ 0 equivalence : {A, B, C, E} {D, F}

1 eq : {A, C} {B, E} {D, F}

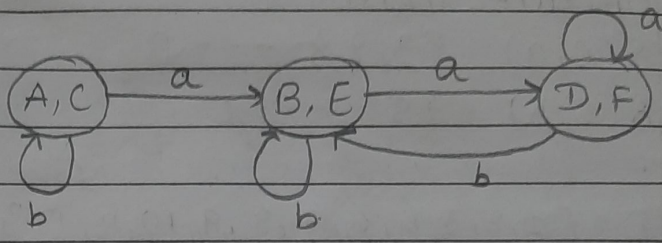
2 eq : {A, C} {B, E} {D, F}

PAGE NO.: 31

Minimized DFA

$\Rightarrow$

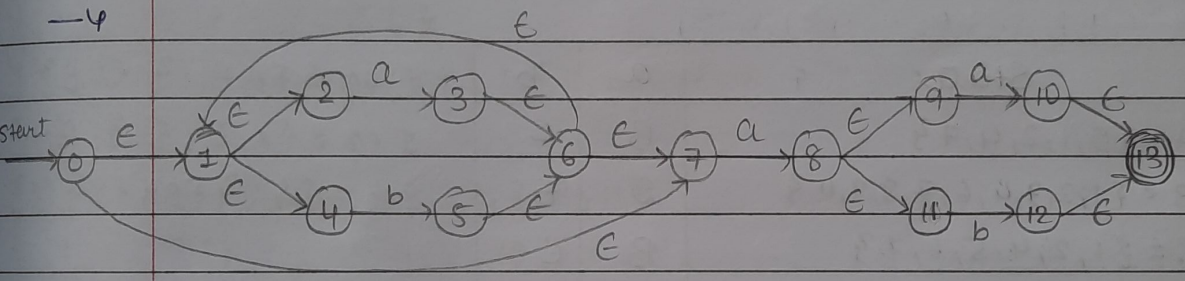
States	a	b
A	B	C
B	D	E
C	E	C
D*	D	E
E	F	E
F*	F	E



Q.17 Construct minimum state DFA's for ~~the~~ following regular expression.

(i) $(a|b)^* a (a|b)$

—4



- ϵ-closure (0) = {0, 1, 2, 4, 7} --- A

$\Rightarrow$ A = {0, 1, 2, 4, 7}

- $\delta(A, a) = \{3, 8\}$

  ϵ-closure (3, 8) = {1, 2, 3, 4, 6, 7, 8, 9, 11} --- B
- $\delta(A, b) = \{5\}$

  ϵ-closure (5) = {1, 2, 4, 5, 6, 7} --- C

$\Rightarrow$ B = {1, 2, 3, 4, 6, 7, 8, 9, 11}

- $\delta(B, a) = \{3, 8, 10\}$

  ϵ-closure (3, 8, 10) = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13} --- D
- $\delta(B, b) = \{5, 12\}$

  ϵ-closure (5, 12) = {1, 2, 4, 5, 6, 7, 12, 13} --- E

⇒ $C = \{1, 2, 4, 5, 6, 7\}$

- $\delta(C, a) = \{3, 8\}$ --- B
- $\delta(C, b) = \{5\}$ --- C

⇒ $D = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13\}$

- $\delta(D, a) = \{3, 8, 10\}$ --- D
- $\delta(D, b) = \{5, 12\}$ --- E

⇒ $E = \{1, 2, 4, 5, 6, 7, 12, 13\}$

- $\delta(E, a) = \{3, 8\}$ --- B
- $\delta(E, b) = \{5\}$ --- C

States	a	b
$A = \{0, 1, 2, 4, 7\}$	B	C
$B = \{1, 2, 3, 4, 6, 7, 8, 9, 11\}$	D	E
$C = \{1, 2, 4, 5, 6, 7\}$	B	C
$D^* = \{1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13\}$	D	E
$E^* = \{1, 2, 4, 5, 6, 7, 12, 13\}$	B	C

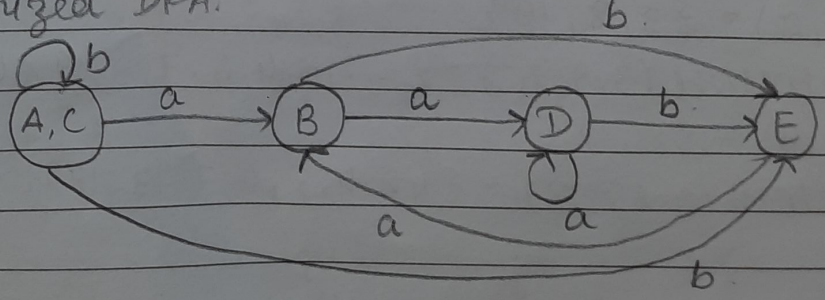
0 eq = {A, B, C} {D, E}

1 eq = {A, C} {B} {D} {E}

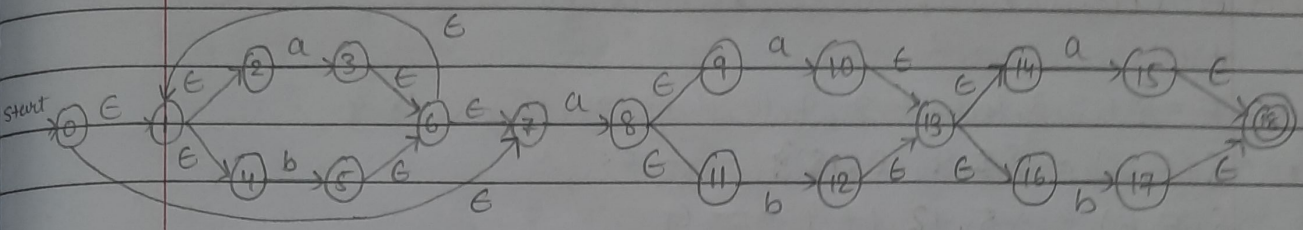
2 eq = {A, C} {B} {D} {E}

state	a	b
{A, C}	{B}	{A, C}
{B}	{D}	{E}
{D}	{D}	{E}
{E}	{B}	{A, C}

⇒ Minimized DFA:



(ii) $(a|b)^* a(a|b)(a|b)$



- ϵ-closure(0) = {0, 1, 2, 4, 7} --- A

⇒ A = {0, 1, 2, 4, 7}

- $\delta(A, a)$ = {3, 8}
  ϵ-closure(3, 8) = {1, 2, 3, 4, 6, 7, 8, 9, 11} --- B
- $\delta(A, b)$ = {5}
  ϵ-closure(5) = {1, 2, 4, 5, 6, 7} --- C

⇒ B = {1, 2, 3, 4, 6, 7, 8, 9, 11}

- $\delta(B, a)$ = {3, 8, 10}
  ϵ-closure(3, 8, 10) = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 16} --- D
- $\delta(B, b)$ = {5, 12}
  ϵ-closure(5, 12) = {1, 2, 4, 5, 6, 7, 12, 13, 14, 16} --- E

⇒ C = {1, 2, 4, 5, 6, 7}

- $\delta(C, a)$ = {3, 8} --- B
- $\delta(C, b)$ = {5} --- C

⇒ D = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 16}

- $\delta(D, a)$ = {3, 8, 10, 15} --- F
  ϵ-closure(3, 8, 10, 15) = {1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 18}
- $\delta(D, b)$ = {5, 12, 17}
  ϵ-closure(5, 12, 17) = {1, 2, 4, 5, 6, 7, 12, 13, 14, 16, 17, 18} --- G

⇒ E = {1, 2, 4, 5, 6, 7, 12, 13, 14, 16}

- $\delta(E, a)$ = {3, 8, 15}
  ϵ-closure = {1, 2, 3, 4, 6, 7, 8, 9, 11, 15, 18} --- H

- $\delta(E,b) = \{5,17\}$

  ϵ-closure $(5,17) = \{1,2,4,5,6,7,17,18\}$ --- I

$\Rightarrow$ $F = \{1,2,3,4,6,7,8,9,10,11,13,14,15,16,18\}$

- $\delta(F,a) = \{3,8,10,15\}$ --- F
- $\delta(F,b) = \{5,12,17\}$ --- G

$\Rightarrow$ $G = \{1,2,4,5,6,7,12,13,14,16,17,18\}$

- $\delta(G,a) = \{3,8,15\}$ --- H
- $\delta(G,b) = \{5,17\}$ --- I

$\Rightarrow$ $H = \{1,2,3,4,6,7,8,9,11,15,18\}$

- $\delta(H,a) = \{3,8,10\}$ --- D
- $\delta(H,b) = \{5,12\}$ --- E

$\Rightarrow I = \{1,2,4,5,6,7,17,18\}$

- $\delta(I,a) = \{3,8\}$ --- B
- $\delta(I,b) = \{5\}$ --- C

States	a	b
A	B	C
B	D	E
C	B	C
D	F	G
E	H	I
F*	F	G
G*	H	I
H*	D	E
I*	B	C

0 eq = {A,B,C,D,E} {F,G,H,I}

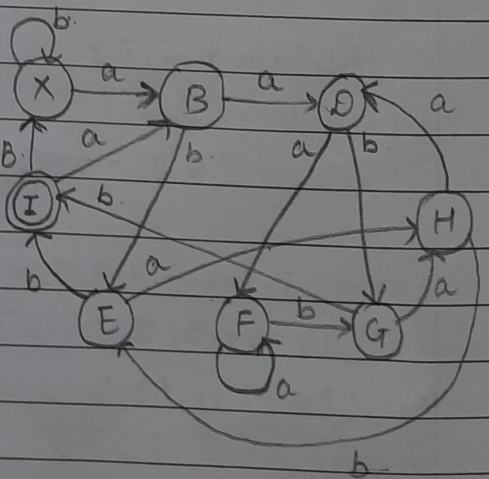
1 eq = {A,B,C} {D,E} {F,G} {H,I}

2 eq = {A,C} {B} {D} {E} {F} {G} {H} {I}

3 eq = {A,C} {B} {D} {E} {F} {G} {H} {I}

Minimized DFA

$\Rightarrow$



States	a	b
{A,C} $\Rightarrow$ {X}	{B}	{X}
{B}	{D}	{E}
{D}	{F}	{G}
{E}	{H}	{I}
{F}*	{F}	{G}
{G}*	{H}	{I}

{H}*	{D}	{F}
{I}	{B}	{X}

Q.18 Explain Minimization of DFA.

Ans Minimization of DFA (Deterministic Finite Automata) is the process of reducing the number of states in a DFA while preserving its language recognition capability. The goal is to create an equivalent DFA that is similar and more efficient in terms of state transitions.

- Steps for DFA Minimization :-

i) Remove Unreachable states : Any state that is not reachable from the initial state is unnecessary and can be removed.

ii) Partition the States : Partition the set of states into equivalence classes. Two states are considered equivalent if :
  - They are in the same group.
  - They behave identically for all input symbols.

iii) Merge Equivalent States : Combine all states ~~while~~ within the same equivalence class into a single state. This reduces the total number of states.

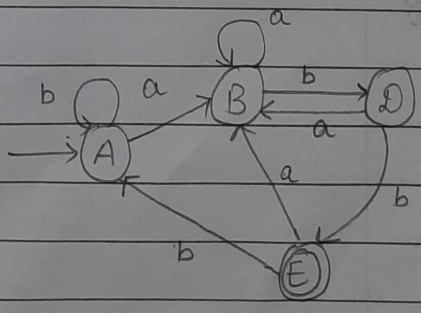
eg!

δ	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E*	B	C

Minimize ⇒

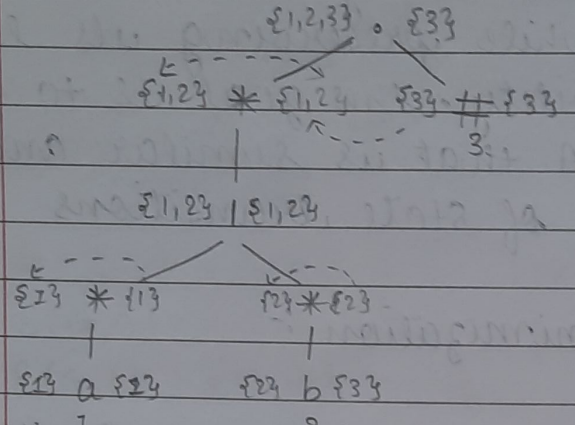
δ	a	b
A	B	A
B	B	D
D	B	E
E*	B	A

⇒



Q.19 Construct DFA for following Regular Expressions. Use Firstpos, Lastpos and Followpos functions to construct DFA: $(a^* | b^*)^*$

→



Position	Followpos
3	–
2	{1,2,3}
1	{1,2,3}

- Initial State = {1,2,3} --- A

⇒ A = {1,2,3}

- δ(A, a) = followpos(1) = {1,2,3} --- A
- δ(A, b) = followpos(2) = {1,2,3} --- A

State	a	b
A	A	A

DFA ⇒