VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JNANA SANGAMA", BELAGAVI – 590018

Automata Theory and Computability (18CS54)

Assignment 1

Submitted by

KARTHIK J 4SF20IS041

In partial fulfillment of the requirements for V Semester of

BACHELOR OF ENGINEERING
IN
INFORMATION SCIENCE & ENGINEERING

Under the Guidance of

Mr. Rithesh Pakkala P.
Assistant Professor
Department of Information Science & Engineering

SAHYADRI
College of Engineering & Management
Mangaluru – 575 007

SAHYADRI

College of Engineering & Management

Mangaluru – 575 007

Department of Information Science & Engineering

CERTIFICATE

This is to certify that the **Assignment 1** on "**Automata Theory and Computability (18CS54)**" has been completed by **KARTHIK J (4SF20IS041)**, the Bonafide student of **Sahyadri College of Engineering & Management** in partial fulfillment for the V semester of Bachelor of Engineering in **Information Science & Engineering** of Visvesvaraya Technological University, Belagavi during the Academic Year 2022 - 23.

Faculty In charge

Mr. Rithesh Pakkala P.

Marks Awarded:				
Q. No.	**Step by Step Approach (3 Marks)**	**Program written in any Language (4 Marks)**	**Output Analysis with Different Test Cases (3 Marks)**	**Total Marks (10)**
1				
2				
3				
4				
5				
			Total Marks (50)	

SAHYADRI
COLLEGE OF ENGINEERING & MANAGEMENT
An Autonomous Institution
MANGALURU

Department of Information Science & Engineering

Assignment I - ODD Semester 2022 - 23

Course Title : Automata Theory and Computability		Course Code: 18CS54
Sem / Section: V 'A'	Faculty: Mr. R Pakkala	Max. Marks: 50
Date of Announcement: 27/12/2022	Date of Submission: 05/01/2023	

Note:

- **Answer All the Five Questions**
- **The Assignment document must contain**
  - **Cover page with Name, USN, Course Title, Course Code and Faculty Details**
  - **Step by Step Approach to design DFSM**
  - **Programming Code**
  - **Screenshots of Execution with different test cases**
- **Student can write a program in any language (C/C++/Java/Python/C#/PHP)**

Q. No.	Questions	Marks	Blooms Level	CO No.
1	Illustrate the design in step by step approach and write a program to simulate deterministic finite state machine (DFSM) for accepting the language L = $\{a^n b^m \mid n \bmod 2=0, m \geq 1\}$. Analyze the output with different test cases.	10	CL3	CO1
2.	Illustrate the design in step by step approach and write a program to simulate a DFSM which accept the language L = {w | w ∈ {a, b}* and Na (w) mod 3 = Nb (w) mod 3}. Analyze the output with different test cases.	10	CL3	CO1
3	Illustrate the design in step by step approach and write a program to simulate a DFSM which accept strings that start and end with same character. Analyze the output with different test cases.	10	CL3	CO1
4	Illustrate the design in step by step approach and write a program to simulate a DFSM which accept Binary strings that starts or ends with "01". Analyze the output with different test cases.	10	CL3	CO1
5	Illustrate the design in step by step approach and write a program to simulate a DFSM which accept the language having all 'a' before all 'b'. Analyze the output with different test cases.	10	CL3	CO1

Cognitive Levels of Bloom's Taxonomy

No.	CL1	CL2	CL3	CL4	CL5	CL6
Level	Remember	Understand	Apply	Analyze	Evaluate	Create

Course Outcomes

CO1	Solve the finite state machine problems for different formal languages by discussing the central concepts of Automata Theory.	CL3
CO2	Solve the regular expression and regular grammar problems. Also discuss the proofs of regular languages.	CL3
CO3	Solve the context free grammar and pushdown automata problems for the different formal languages.	CL3
CO4	Discuss the algorithms and decision procedures for context free languages. Also solve the turing machine problems for the different formal languages.	CL3
CO5	Discuss the concepts of decidability and complexity related to computational problems.	CL2

Assessment Method		
Sl. No.	**Assessment Component**	**Marks Allotted**
1.	Step by Step Approach	3
2.	Program written in any Language	4
3.	Output Analysis with Different Test Cases	3

1. Illustrate the design in step by step approach and write a program to simulate deterministic finite state machine (DFSM) for accepting the language L = $\{a^n b^m \mid n \bmod 2=0, m \geq 1\}$. Analyze the output with different test cases.

Steps to design DFSM:

Here Language L is given as

L= {b,aab,aaaab,aaaabb,aaaabbb,aaaaaaaab,…….}

Length of minimum string is 1. Therefore, there must be at least 2 states.

Step 1: Make an initial state 'q0'. The minimum possible string is 'b' which is acceptable. For this, make the transition of 'b' from state 'q0' to state 'q3' and notice this state 'q3' as the final state.

Step 2: Now, we have designed the DFA with at least one 'b' . To accept all the strings that start with even number of 'a' like aab, aaaab, aaaabb,….etc, we need to make a new state 'q1' and make a transition of 'a' from 'q0' to 'q1' and transition of 'a' from 'q1' to 'q2' also make the transition a from 'q2' back to 'q1' in order to accept even number of a's.

Step 3: To get more than one b's we make transition b from 'q2' to 'q3' and give a self-loop transition for the state 'q3'.

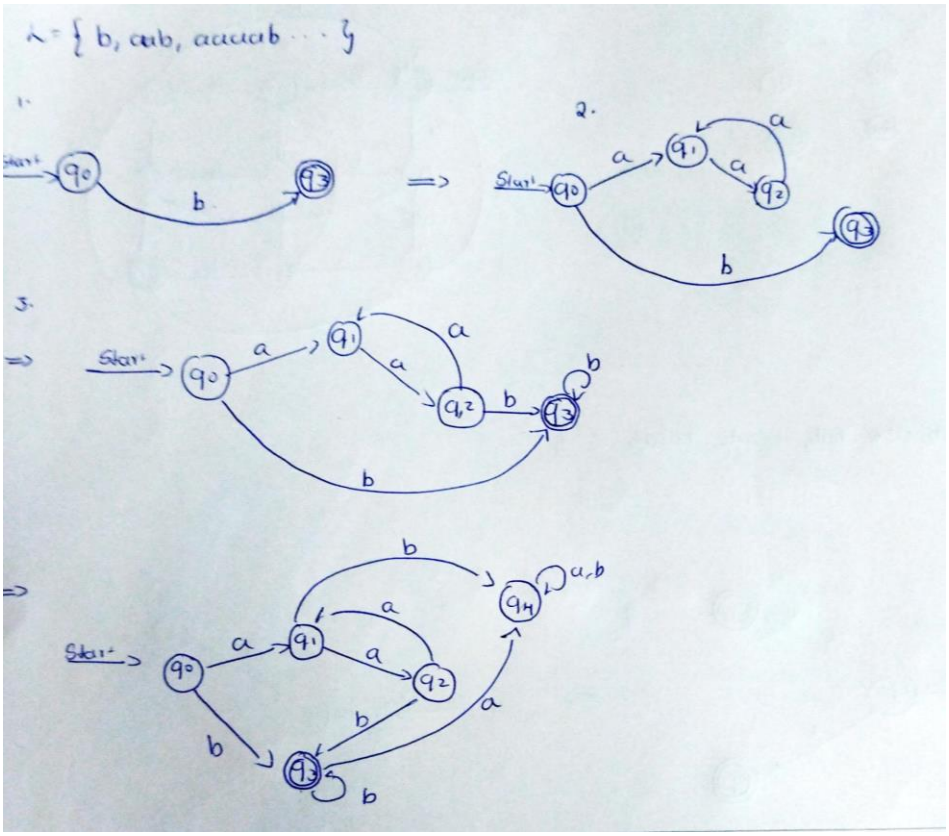
Step 4: The remaining transition from each state are mapped to the dead state 'q4'.

M= ({q0, q1, q2, q3, q4},{a ,b}, δ , q0 , {q3})

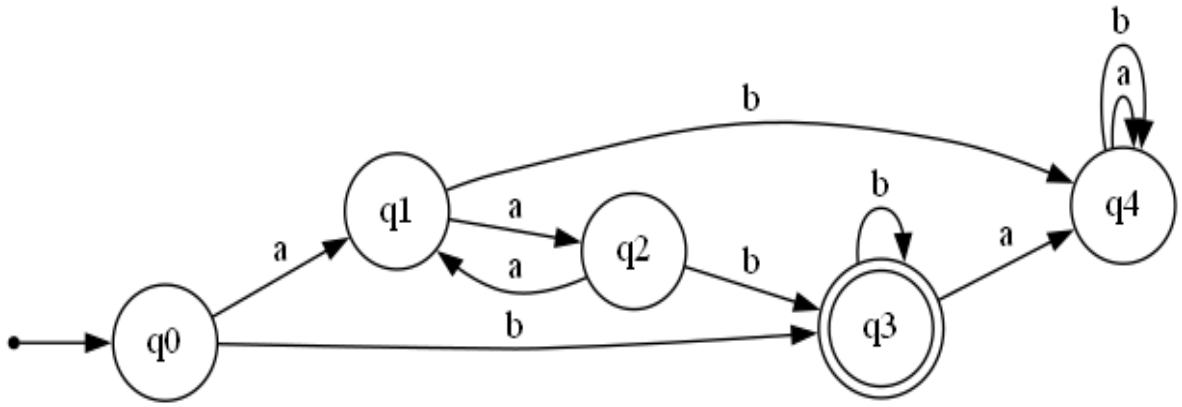
where δ is given by

	δ	a	b
→	q0	q1	q3
	q1	q2	q4
	q2	q1	q3
*	q3	q4	q3
	q4	q4	q4

State 'q3' leads to the acceptance of the string, whereas state 'q4' leads to the rejection of the string.



DFSM:



Program:

```
from automata.fa.dfa import DFA
from visual_automata.fa.dfa import VisualDFA
dfa = VisualDFA(
states={"q0", "q1", "q2", "q3", "q4"},
input_symbols={"a", "b"},
transitions={
"q0": {"a": "q1", "b": "q3"},
```

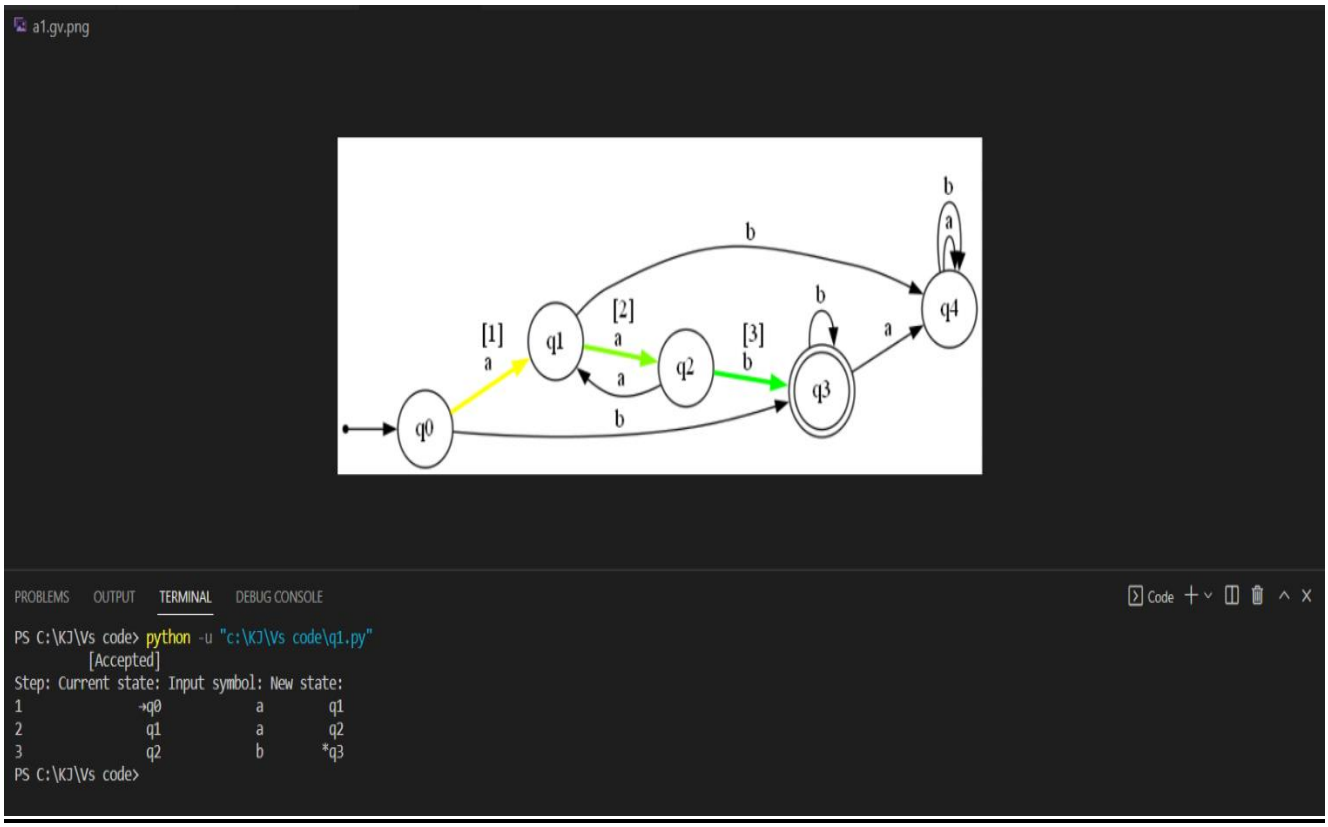
```
"q1": {"a": "q2", "b": "q4"},
"q2": {"a": "q1", "b": "q3"},
"q3": {"a": "q4", "b": "q3"},
"q4": {"a": "q4", "b": "q4"},
},
initial_state="q0",
final_states={"q3"},
)
dfa.show_diagram(view=True,filename="gg10.gv",input_str="")
```

Test Case Analysis:

Output 1:

String: ‘aab’

Accepted

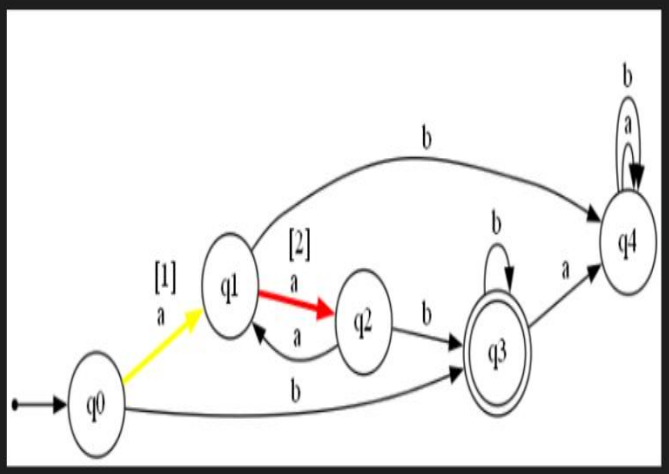


Output 2:

String: ‘aa’

Rejected

a1.gv.png



PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\KJ\Vs code> python -u "c:\KJ\Vs code\q1.py"
        [Rejected]
Step: Current state: Input symbol: New state:
1              →q0          a       q1
2               q1          a       q2
PS C:\KJ\Vs code>
```

2. Illustrate the design in step by step approach and write a program to simulate a DFSM which accept the language L = {w | w ∈ {a, b}* and $N_a(w) \bmod 3 = N_b(w) \bmod 3$}. Analyze the output with different test cases.

Steps to design DFSM:

Here Language L is given as

L= {ε,aaa,bbb,aaabbb,aaaaaabbbbbb,…….}

Step 1: Make a table to determine accepting and rejecting states

$n_a(w)$	$A=n_a(w)\%3$	$n_b(w)$	$B=n_b(w)\%3$	A=B?	State
0	0	0	0	Yes	00
0	0	1	1	No	01
0	0	2	2	No	02
1	1	0	0	No	10
1	1	1	1	Yes	11
1	1	2	2	No	12
2	2	0	0	No	20
2	2	1	1	No	21
2	2	2	2	Yes	22

initial state "00". The minimum possible string is 'ε' which is acceptable. For this, make the start state as accepting state.

Step 2: Wherever there is "yes" marked in the table, make that state as accepting state.

Step 3:

ab ab ab

00 01 02

10 11 12

20 21 22

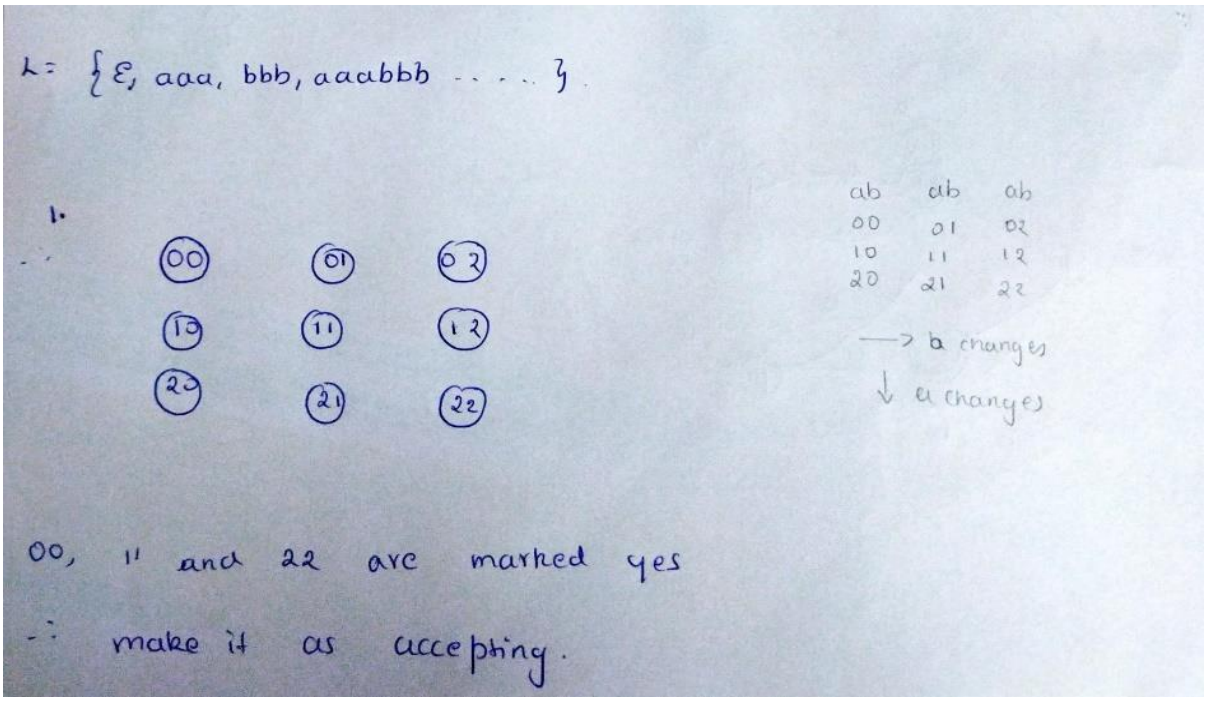
We note that horizontally "b" changes and vertically "a" changes , so we make the transition accordingly.

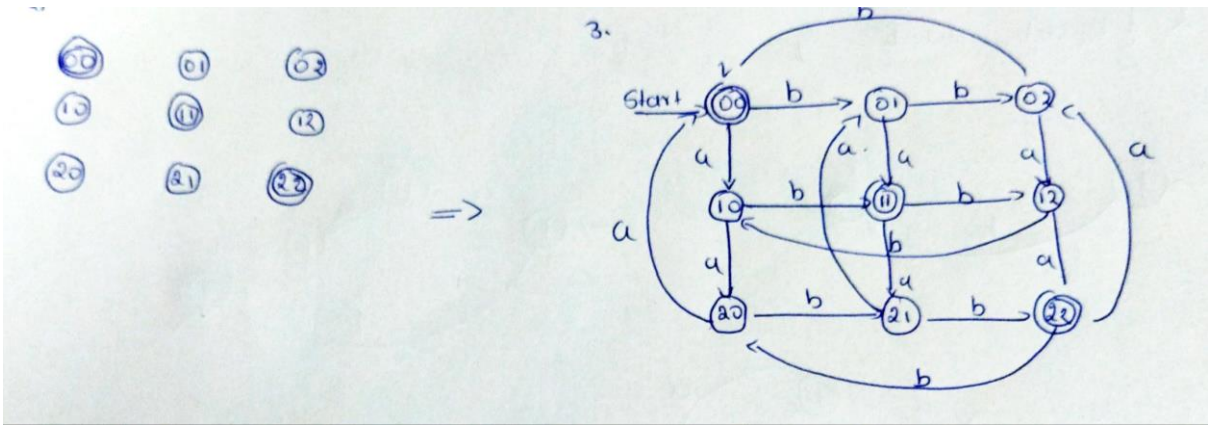
Step 4: M= ({00,01,02,10,11,12,20,21,22}, {a, b}, δ, 00, {00,11,22})

where δ is given by

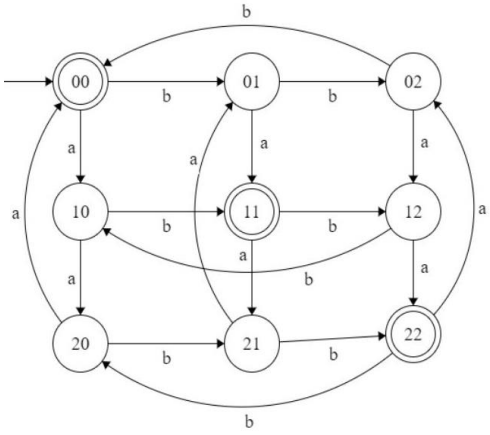
	δ	a	b
→*	00	10	01
	01	11	02
	02	12	00
	10	20	11
*	11	21	12
	12	22	10
	20	00	21
	21	01	22
*	22	02	20

State 00, 11 and 22 leads to the acceptance of the string.





DFSM:



Program:

```
from automata.fa.dfa import DFA
from visual_automata.fa.dfa import VisualDFA
dfa = VisualDFA(
states= {"00", "01", "02", "10", "11","12","20","21","22"},
input_symbols= {"a", "b"},
transitions={
"00": {"a": "10", "b": "01"},
"01": {"a": "11", "b": "02"},
"02": {"a": "12", "b": "00"},
"10": {"a": "20", "b": "11"},
"11": {"a": "21", "b": "12"},
"12": {"a": "22", "b": "10"},
"20": {"a": "00", "b": "21"},
"21": {"a": "01", "b": "22"},
"22": {"a": "02", "b": "20"},
},
initial_state="00",
final_states= {"00","11","22"},
)
dfa.show_diagram(view=True,filename="gg20.gv",input_str="")
```

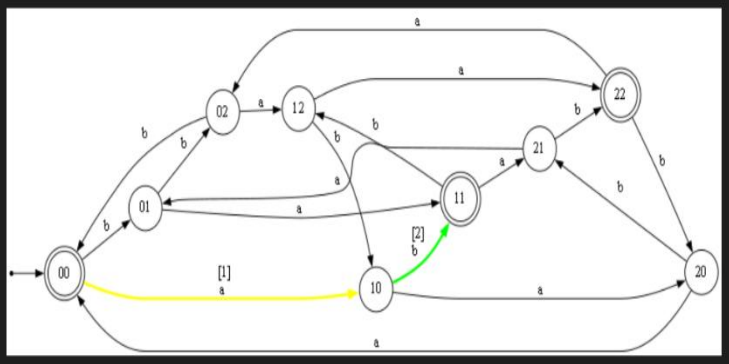
Test Case Analysis:

Output 1:

String: ‘ab’

Accepted

a2.gv.png



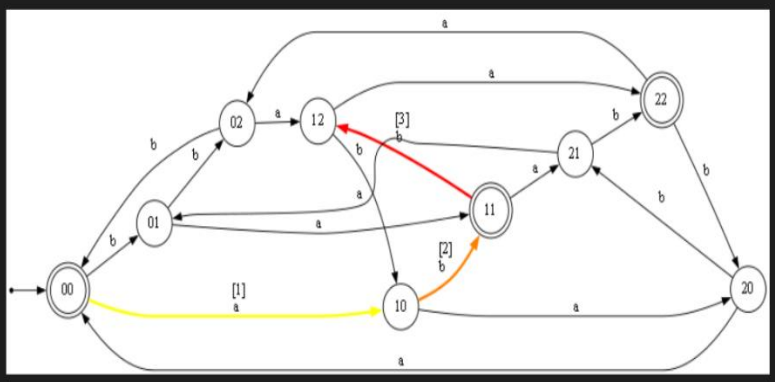

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    Code
PS C:\KJ\Vs code> python -u "c:\KJ\Vs code\q2.py"
        [Accepted]
Step: Current state: Input symbol: New state:
1              →*00           a        10
2               10            b       *11
PS C:\KJ\Vs code>
```

Output 2:

String: ‘abb’

Rejected

a2.gv.png




```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                    Code
PS C:\KJ\Vs code> python -u "c:\KJ\Vs code\q2.py"
        [Rejected]
Step: Current state: Input symbol: New state:
1              →*00           a        10
2               10            b       *11
3              *11            b        12
PS C:\KJ\Vs code>
```

3. Illustrate the design in step by step approach and write a program to simulate a DFSM which accept strings that start and end with same character. Analyze the output with different test cases.

Steps to design DFSM:

Here Language L is given as

L= {a,b,aba,bab,ababa,babab,abbba,baaab,…….}

Step 1: Make an initial state "q0". The minimum possible string is 'a' or 'b' which is acceptable. Create new states q1 and q3 and make it accepting. Give the transition of "a" from q0 to q1 and transition of "b" from q0 to q3.

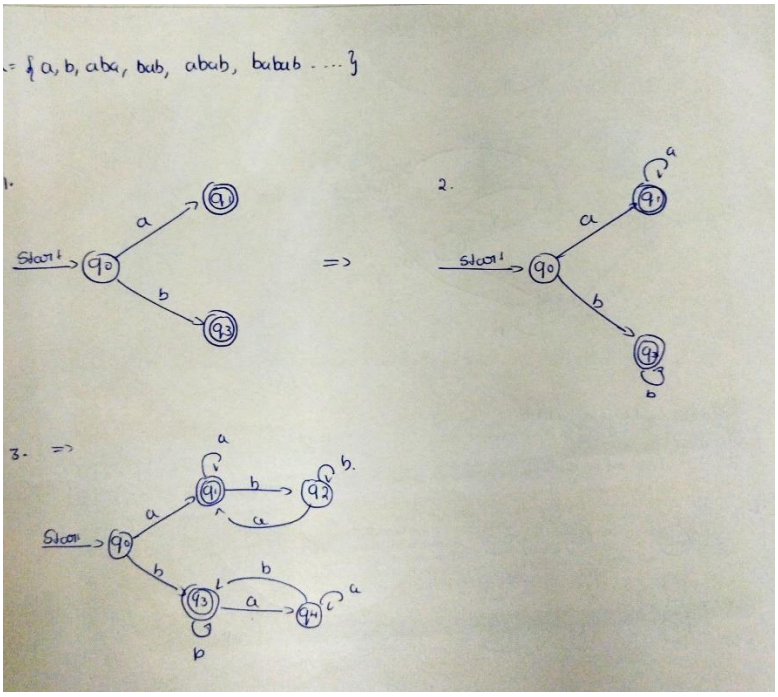
Step 2: We define the remaining transition by giving self loop of a for state q1 and b for q3.

Step 3: Create new states q2 and q4. Make the transitions in such a way that the start and the final symbols are same.

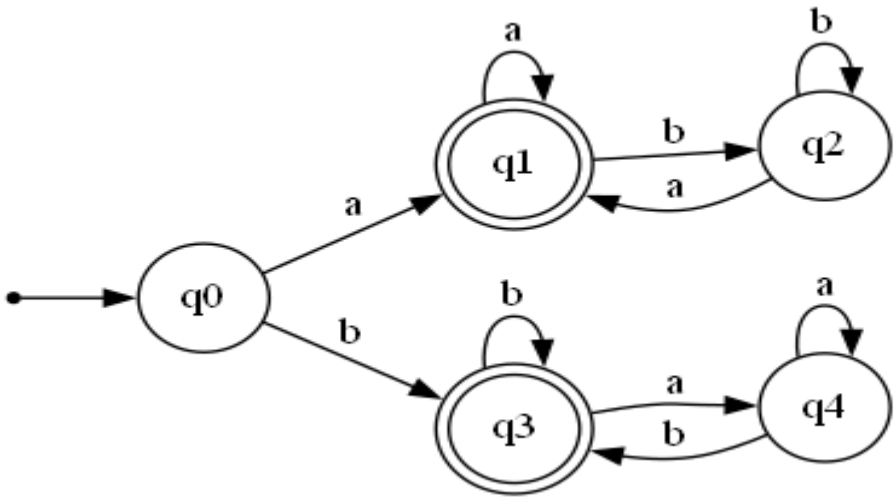
Step 4: M= ({q0, q1, q2, q3, q4},{a ,b}, δ , q0, {q1, q3})

where δ is given by

	δ	a	b
→	q0	q1	q3
*	q1	q1	q2
	q2	q1	q2
*	q3	q4	q3
	q4	q4	q3



DFSM:



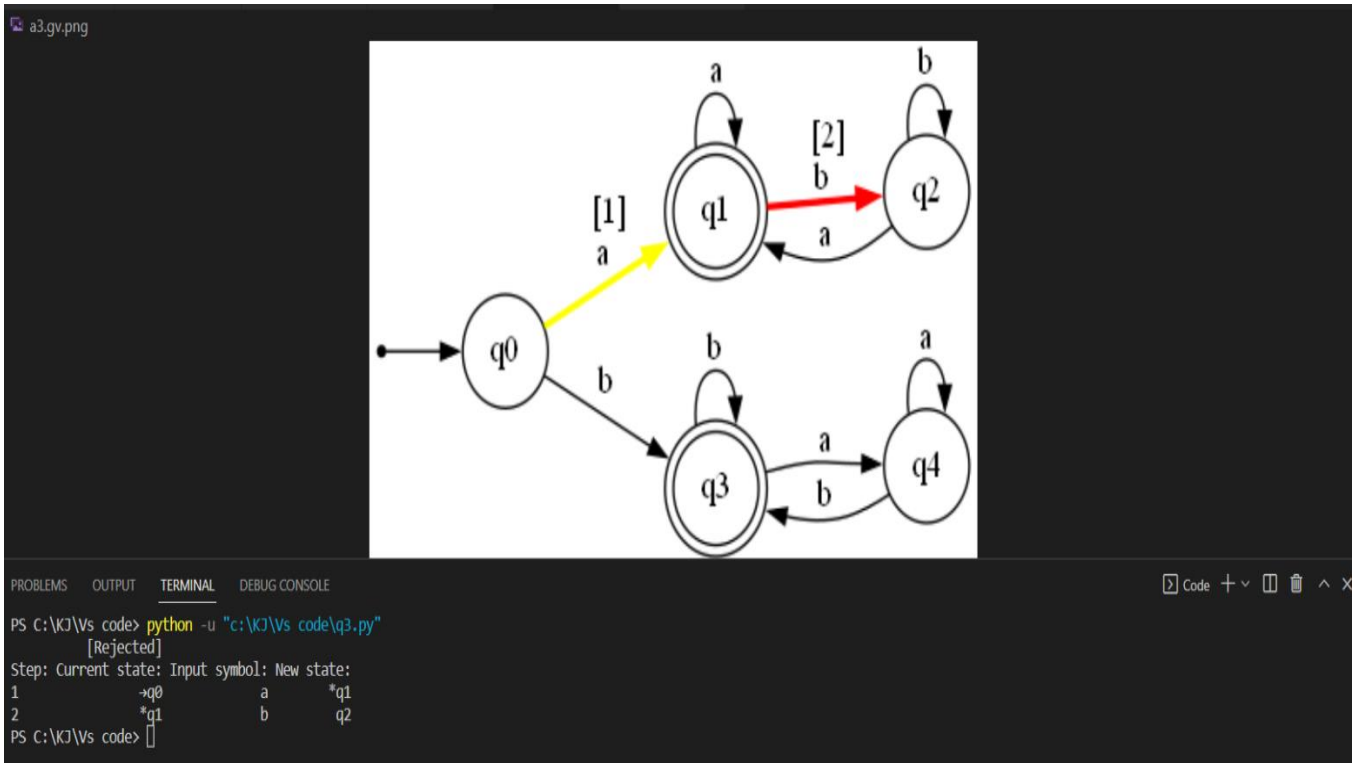
Program:

```
from automata.fa.dfa import DFA
from visual_automata.fa.dfa import VisualDFA
dfa = VisualDFA(
states={"q0", "q1", "q2", "q3", "q4"},
input_symbols={"a", "b"},
transitions={
"q0": {"a": "q1", "b": "q3"},
"q1": {"a": "q1", "b": "q2"},
"q2": {"a": "q1", "b": "q2"},
"q3": {"a": "q4", "b": "q3"},
"q4": {"a": "q4", "b": "q3"},
},
initial_state="q0",
final_states={"q1","q3"},
)
dfa.show_diagram(view=True,filename="gg30.gv",input_str=" ")
```

Test Case Analysis:

Output 1:

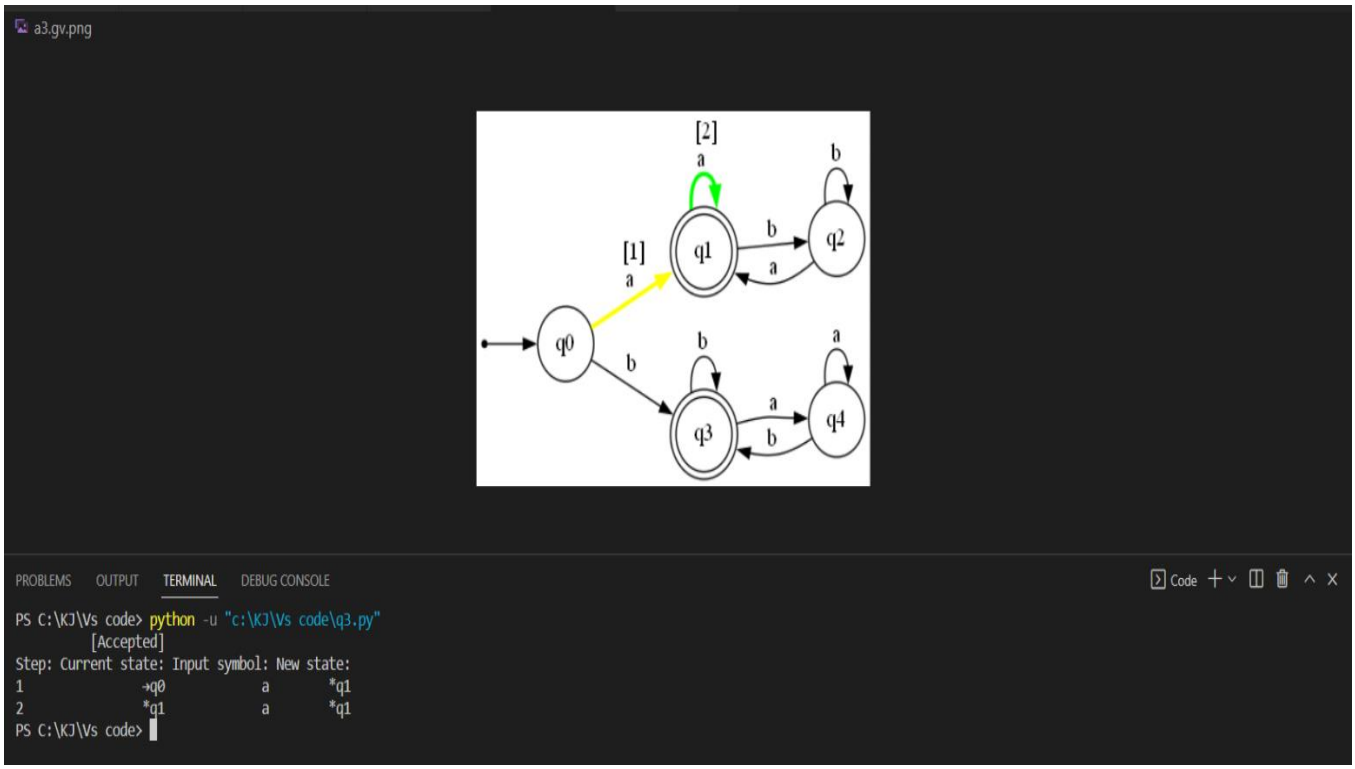
String: ‘ab’
Rejected



Output 2:

String: ‘aa’

Accepted



4. Illustrate the design in step by step approach and write a program to simulate a DFSM which accept Binary strings that starts or ends with “01”. Analyze the output with different test cases.

Steps to design DFSM:

Here Language L is given as

L= {01,001,101,0001,0101,1001,1101,…….}

Step 1: Make an initial state “q0”. Next we do nesting one for starting with “01” another one for ending with “01”.

Step 2: Starting with “01” it can also end with “01” with minimum string “01” so we make “q2” as accepting state followed by any number of 0’s and 1’s.

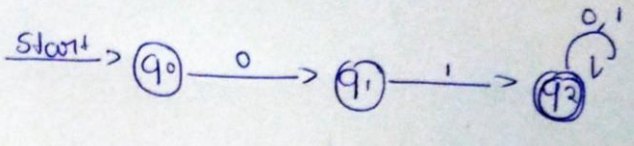
Step 3: Ending with ‘01’, we define the transitions which can have any number of 0’s and 1’s and finally ends with ‘01’. We make “q5” as the accepting state. Finally we club both the transitions.

Step 4: M= ({q0, q1, q2, q3, q4,q5},{0 ,1}, δ , q0, { q2, q5})

where δ is given by

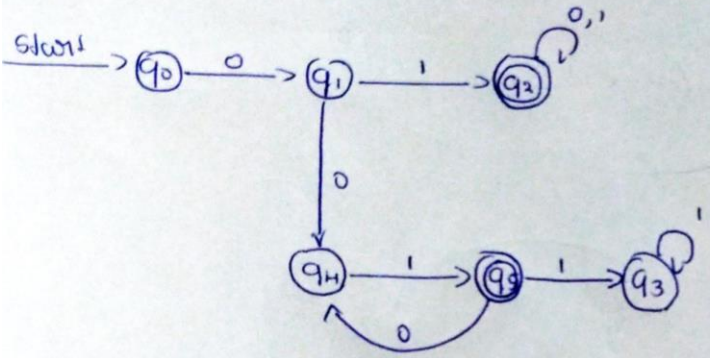
	δ	a	b
→	q0	q1	q3
	q1	q4	q2
*	q2	q2	q2
	q3	q4	q3
	q4	q4	q5
*	q5	q4	q3

$L = \{01, 001, 101, 0001, 0101, 1001, 1101 \ldots\}$

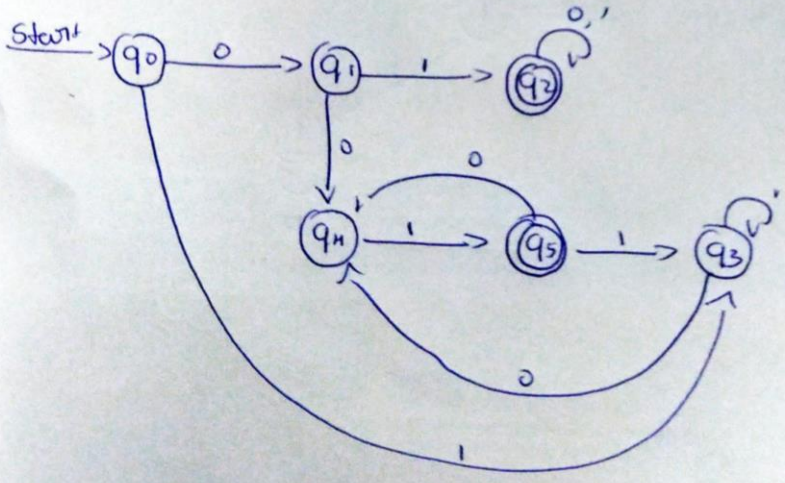


(Starts with 01)

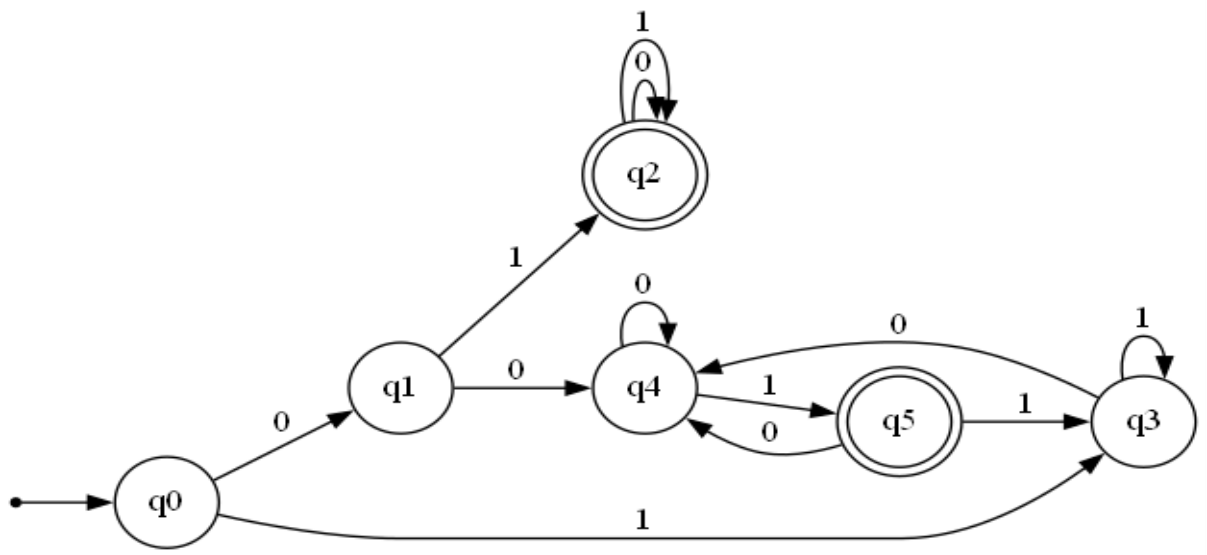
=>



=>



DFSM:



Program:

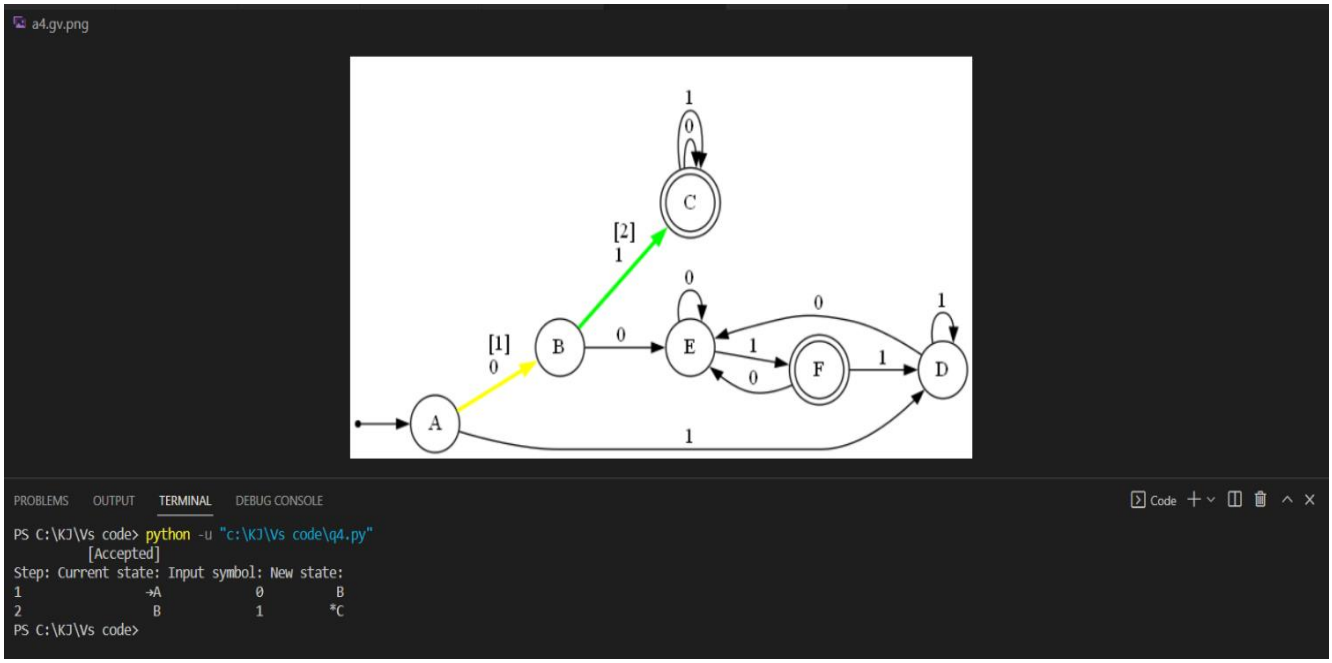
```
from automata.fa.dfa import DFA
from visual_automata.fa.dfa import VisualDFA
dfa = VisualDFA(
states={"q0", "q1", "q2", "q3", "q4","q5"},
input_symbols={"0", "1"},
transitions={
"q0": {"0": "q1", "1": "q3"},
"q1": {"0": "q4", "1": "q2"},
"q2": {"0": "q2", "1": "q2"},
"q3": {"0": "q4", "1": "q3"},
"q4": {"0": "q4", "1": "q5"},
"q5": {"0": "q4", "1": "q3"},
},
initial_state="q0",
final_states={"q2","q5"},
)
dfa.show_diagram(view=True,filename="gg40.gv",input_str="")
```

Test Case Analysis:

Output 1:

String: ‘01’

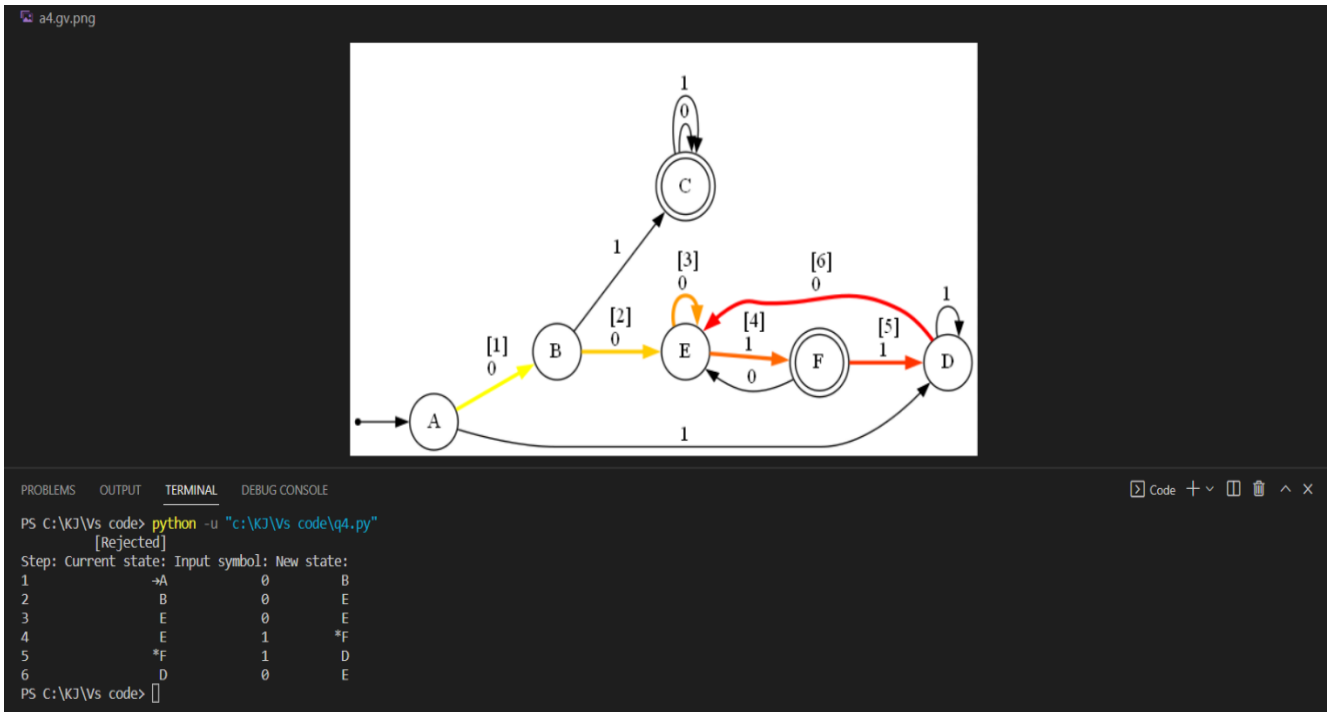
Accepted



Output 2:

String: '000110'

Rejected



5. Illustrate the design in step by step approach and write a program to simulate a DFSM which accept the language having all ‘a’ before all ‘b’. Analyze the output with different test cases.

Steps to design DFSM:

Here Language L is given as

L= {a,b,aab,aaaab,aaaabb,aaaabbb,aaaaaaaab,…….}

Step 1: Make an initial state “q0”. Define the transition ‘a’ from “q0” to “q1”. Similarly, define for ‘b’ from “q0” to “q2”. Make “q1” and “q2” as accepting.

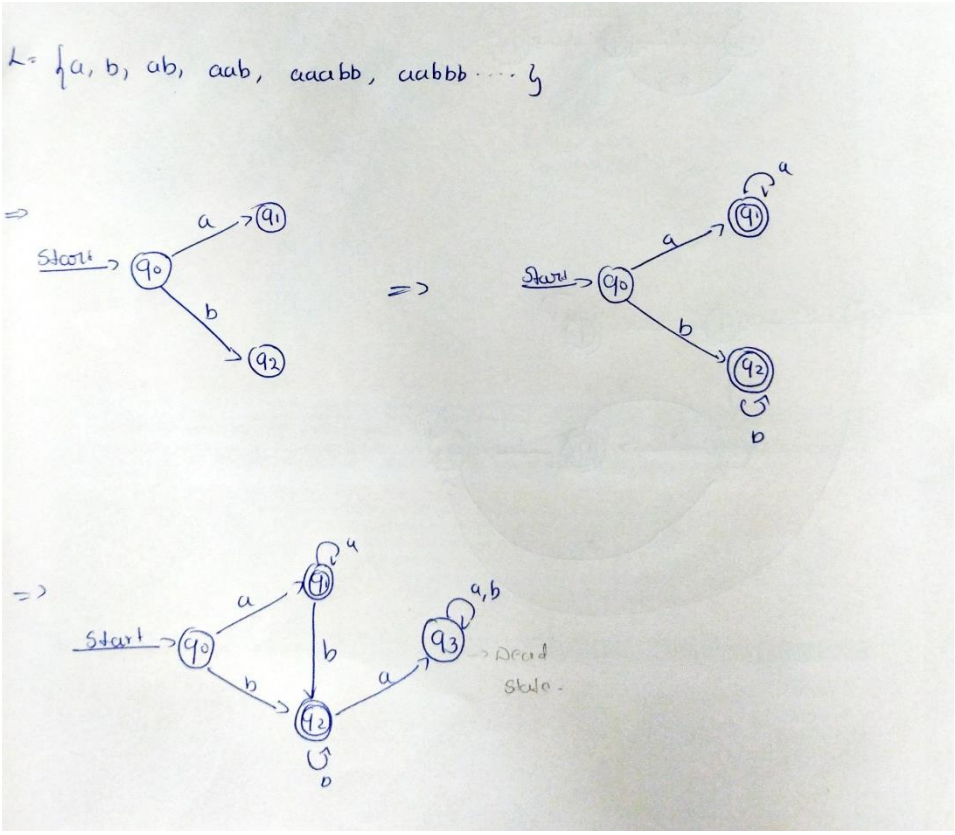
Step 2: To get multiple a’s and b’s give loops to ‘q1’ and ‘q2’ respectively. Give the transition ‘b’ from ‘q1’ to ‘q2’.

Step 3: Give remaining transitions from ‘q2’ to the dead state ‘q3’.

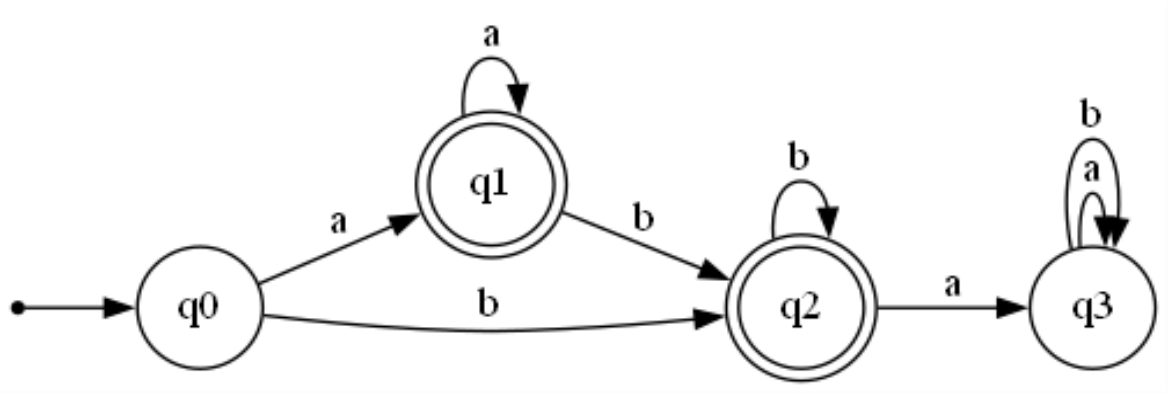
Step 4: M= ({q0, q1, q2, q3},{a ,b}, δ , q0, {q1,q2})

where δ is given by

	δ	a	b
→	q0	q1	q2
*	q1	q1	q2
*	q2	q3	q2
	q3	q3	q3



DFSM:



Program:

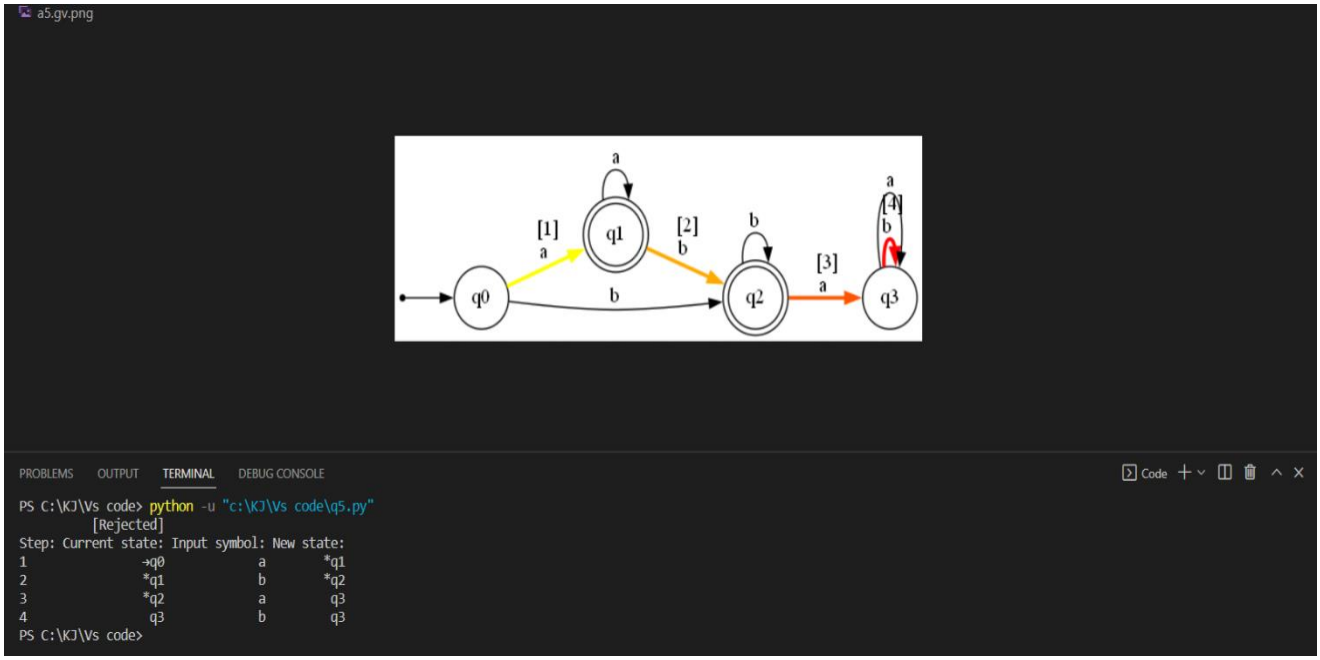
```
from automata.fa.dfa import DFA
from visual_automata.fa.dfa import VisualDFA
dfa = VisualDFA(
states={"q0", "q1", "q2", "q3"},
input_symbols={"a", "b"},
transitions={
"q0": {"a": "q1", "b": "q2"},
"q1": {"a": "q1", "b": "q2"},
"q2": {"a": "q3", "b": "q2"},
"q3": {"a": "q3", "b": "q3"},

},
initial_state="q0",
final_states={"q1","q2"},
)
dfa.show_diagram(view=True,filename="gg50.gv",input_str="")
```

Test Case Analysis:

Output 1:

String: ‘abab’
Rejected



Output 2:

String: ‘aaabbb’

Accepted