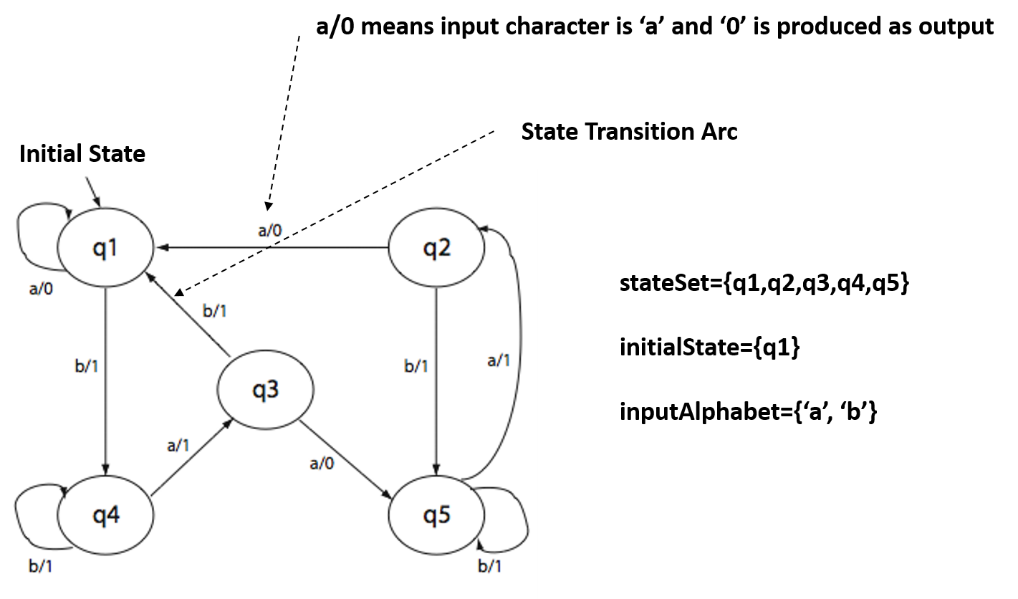
Q2. A **Finite State Machine (FSM)** is represented by a tuple of the form **(inputAplhabets, stateSet, initialState, currentState, currentOutput and stateTransitionFunction).** Where ‘**inputAplhabets’** is a set of characters that a FSM takes as inputs, **‘stateSet’** is the set of states represented by a given FSM, ‘**initialState’** represents the initial state of the FSM, **‘currentState’** represents the current state of the FSM, **‘currentOutput’** represents the current output produced by the FSM depending upon the supplied input character and ‘**stateTransitionFunction’** represents the transition function according to which a FSM makes transition in different states.



**Figure Q2.**

An example FSM is shown in Figure Q2. This FSM has five different states namely {**q1, q2, q3, q4, q5**}. The state ‘**q1**’ is the initial state and this FSM takes only two characters {either **‘a’** or **‘b’**} as inputs in any of the five states and produces either ‘0’ or ‘1’ output. The next state transition of the FSM may be same or a different from the previous state.

The Figure Q2 also shows

1. How a FSM makes transitions in different states and
2. What characters are produced as outputs in each state?

For example, in state **‘q1’** if character **‘a’** is given as an input then **‘0’** will be produced as output and the state **’q1’** will be the next current state. Similarly, in state **‘q5’** if **‘a’** is given as an input then **‘1’** will be produced as output and the state **‘q2’** will be the next current state of the FSM. The same explanation is applicable for all state transitions shown in the Figure Q2. Initially, the values of both **‘initialState’** and **‘currentState’** attributes are same.

You have to implement a class named ***FSM***, which encapsulates the features of a FSM as described above and as shown in Figure Q2. You are given below an incomplete Java code of ***FSM*** class and you have to complete it as per the commented specification given in the class itself. Assume all relevant packages are imported. [Note read the given FSM code carefully before the attempt]

**class FSM**

**{**

**private char[] inputAplhabets; // input characters of the FSM**

**private String[] stateSet; // states of the FSM**

**private String initialState; // initial state of the machine**

**private String currentState; // current state of the machine**

**private char currentOutput; // current output of the machine**

**// Constructor Method**

**FSM(char[] inputAplhabets, String[] stateSet, String initialState)**

**{**

**this.inputAplhabets = inputAplhabets; this.stateSet=stateSet;**

**this.initialState=initialState; currentState=initialState;**

**cuurentOutput=’’;**

**}// End of Constructor Method**

**// Accessor Methods**

**public char[] getInputAplhabets() {return this. inputAplhabets;}**

**public String[] getStateSet() { return this.stateSet;}**

**public String getInitialState() { return this.initialState;}**

**public String getCurrentState() { return this.currentState;}**

**public char getCurrentOutput() { return currentOutput; }**

**// Method to reset the machine to its initial state**

**public void resetMachine()**

**{**

**currentState = initialState;**

**currentOutput = ’’;**

**}**

**public String[] getInputStringsofLength(char[] inputAlphabets, int lengthK)**

**{**

/\* This method returns all the strings of lengthK formed from the characters of inputAlphabets. If the values passed for inputAlphabets and lengthK parameters are {‘a’,’b’} and 1 respectively, then it returns only two strings as “a” and “b”. [Because only two strings of length 1 are possible for this case]. If the lengthK parameter value is changed to ‘2’ then this method will return four strings as {“aa”,”ab”,”ba”, “bb”}. \*/

// ASSUME THE CODE OF THIS METHOD IS AVAILABLE FOR USE

**}**

**public void stateTransitionFunction(char inputC)**

**{**

**Q.2.1** /\* Complete this method as per following specification \*/

/\* This method sets the currentState and currentOuput attributes of the FSM as shown in Figure Q2. Note you have to write this function as per the state transition pattern as shown in Figure Q2\*/

switch (inputC)

{

case ‘a’:

// Write your code here

case ‘b’:

// Write your code here

**}**// End of switch statement

**}**// End of method stateTransitionFunction

**public String getBinaryStringOutput(String inputStr)**

**{**

**Q.2.2** /\*Complete this method as per the following specification \*/

/\* This method returns the output produced by the FSM of Figure Q2 when ‘inputStr’ is supplied as an input when the values of both ‘currentState’ and ‘initialState’ attributes are same. For example, if value of ‘inputStr’ is “abb” then this method returns the string “011” as an output. Note that from the input string “abb”, the first character ‘a’ is used as an input in state “q1” (initial state) and ‘0’ is produced as output. The next character ‘b’ is used as input for finding out subsequent outputs and this process is repeated until the last input character of ‘inputStr’.

Similarly if value of ‘inputStr’ is “babbaab” then output string return by this method is “1111101” \*/

**String binaryString=””;**

**if(inputStr.length() == 0) return binaryString;**

**// Write your code from here**

**return binaryString;**

**}**// End of method getBinaryStringOutput

**public int getMachineEquivalent(FSM other)**

**{**

**Q.2.3** /\* Complete this method as per following specification \*/

/\* This method returns an integer named ‘equivalent’ (<=20) that represents the highest length of any strings for which the invoking **FSM** and the ‘other’ FSM will produce the identical outputs.

For example, if the statement f1.getMachineEquivalent(f2) returns a value 5, then it means the FSMs ‘f1’ and ‘f2’ will produce identical outputs for any input string of length 5. Note that if FSMs ‘f1’ and ‘f2’ returns identical outputs for any string of length 5 then they will also produce identical outputs for any string having lengths from 1 to 4 also. The outputs of FSMs ‘f1’ and ‘f2’ will be different for any input string of length greater than 5.

Assume that the invoking FSM and the ‘other’ FSM references have similar values for ‘inputAlphbets’ and ‘stateSet’ attributes.

\*/

**int equivalent = 0;**

**// Write your code from here**

**return equivalent;**

**}**// End of method getMachineEquivalent

**}// End of class FSM**

**[12+12+16=40M]**